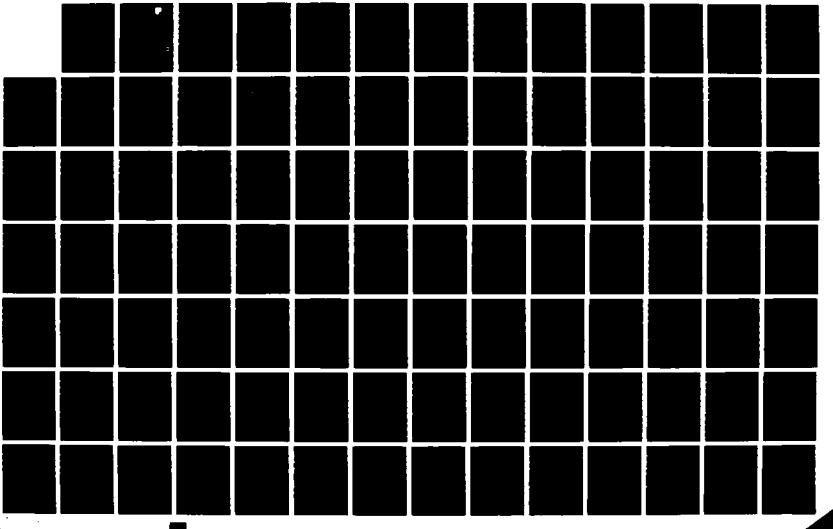
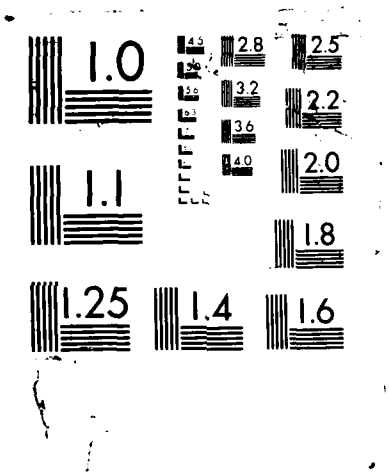
AD-A190 019 METHODOLOGY FOR SOFTWARE RELIABILITY PREDICTION VOLUME 2(U) SCIENCE APPLICATIONS INTERNATIONAL CORP SAN DIEGO CA J MCCALL ET AL. NOV 87 RADC-TR-87-171-VOL-2 1/2

UNCLASSIFIED F30602-83-C-0110 F/G 12/5 NL




1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8

(4)

AD-A190 019

RADC-TR-87-171, Vol II (of two)
Final Technical Report
November 1987

METHODOLOGY FOR SOFTWARE RELIABILITY PREDICTION

Science Applications International Corporation

J. McCall, W. Randall, C. Bowen, N. McKelvey, R. Senn, J. Morris, H. Hecht, S. Fenwick, P. Yates, M. Hecht and R. Vienneau

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-87-171, Vol II (of two) has been reviewed and is approved for publication.

APPROVED: Joseph P. Cavano, Jr

JOSEPH P. CAVANO
Project Engineer

APPROVED: Raymond P. Urtz Jr

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: Richard W. Pouliot

RICHARD W. POULIOT
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COEE) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-87-171, Vol II (of two)
6a. NAME OF PERFORMING ORGANIZATION Science Applications International Corporation 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COEE)
6c. ADDRESS (City, State, and ZIP Code) 10260 Campus Point Drive San Diego CA 92121	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center 8b. OFFICE SYMBOL (If applicable) COEE	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-83-C-0118
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 62702F; PROJECT NO: 5581; TASK NO: 20; WORK UNIT ACCESSION NO: 43

11. TITLE (Include Security Classification)
METHODOLOGY FOR SOFTWARE RELIABILITY PREDICTION

12. PERSONAL AUTHOR(S) J. McCall, W. Randall, C. Bowen, N. McKelvey, R. Senn, J. Morris, H. Hecht, S. Fenwick, P. Yates, M. Hecht, R. Vienneau

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM Jun 83 TO May 87	November 1987	180

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
12	05		Software Reliability
12	08		Software Reliability Engineering
			Software Measurement

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Guidebook provides detailed procedures for the preparation of software reliability predictions and estimations on DOD projects. In developing the Guidebook, 59 software systems were examined and 19 key variables were identified that affected the software reliability of those systems. Procedures to measure these variables were developed to account for the type of application, development, environment, various software characteristics (such as modularity and complexity), test technique, test effort and test coverage. A methodology was also provided to use these measures to predict software fault density and software failure rates.

The Guidebook could be applied by an Air Force acquisition office to help plan for adequate software reliability early in a project's life, specify achievable software reliability goals in a RFP, evaluate progress toward those goals at key project milestones and decide when to release the software. The Guidebook could also be used by the technical staff to establish thresholds for critical measures such as complexity.

(Over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Joseph P. Cavano	22b. TELEPHONE (Include Area Code) (315) 330-4063	22c. OFFICE SYMBOL RADC (COEE)

DD Form 1473, JUN 86 *Previous editions are obsolete*

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

In addition, the Guidebook also contains Quality Review and Standards Review Checklists that can be used in conjunction with the software reliability prediction and estimation methodology. The Quality Review Checklists are used to assess the quality of the requirements and design representation of the software while the Standards Review Checklist would be applied to software code. The checklists provide good guidance for ensuring that quality is built into the software.

PREFACE

Producing more reliable software will lower both production costs and post deployment support costs because additional resources will not be expended correcting problems that could have been avoided or detected earlier. If the most error-prone projects could just be brought up to today's average, their fault density could be cut 2 to 4 times. To make this level of improvement possible, RADC developed this guidebook to improve software reliability. Air Force acquisition offices could apply the guidebook to specify achievable and measurable reliability goals in terms of fault density and failure rate and evaluate progress toward those goals at key project milestones.

To develop the software reliability prediction and estimation methodology described in this guidebook, RADC analyzed 59 projects, totaling over 5 million lines of code. Although the 59 projects studied do not represent all the diversity of Air Force applications, the results should be useful to any project with high reliability requirements that can be matched to the generic applications used here. As RADC receives additional reliability information based on this methodology--it is hoped that project experience will be shared with RADC/COEE on a confidential basis--this guidebook will be updated.

In the absence of quantitative data dealing with software reliability concepts, the RADC guidebook provides information, based not on conjuncture but on measurement, to help predict and estimate fault density and failure rate. The real value of this guidebook is not the numbers it will produce--since they are only the means to the end--but rather the processes it promotes for planning software reliability and controlling key variables that are shown to affect reliability.

For example, there is empirical evidence that a software component's fault density doubles when its complexity exceeds certain thresholds. Although it is not known that increased complexity causes more errors to be introduced, any component exceeding the established threshold should be carefully reviewed or modified to reduce its complexity.

To more actively promote software quality engineering, the guidebook contains checklists that can be used in conjunction with the software reliability prediction and estimation methodology. The Quality Review Checklist represents important quality questions to be answered from requirement and design documentation; the Standards Review Checklist applies to software code. The checklists provide guidance for ensuring that quality is engineered into the software during its development.

To support the data collection inherent in this approach, RADC developed the Automated Measurement System (AMS); the AMS is a quality analysis tool that reduces the cost to collect, store, and analyze quality-related data. The AMS interfaces to requirements and design tools to support early life cycle quality analyses and also analyzes Ada and Fortran code. The AMS runs under VAX/VMS as well as a MicroVax II Workstation. A beta-test version of the AMS is currently available from RADC/COEE to support quality analysis on Defense Dept projects.

Joseph P. Cavano
Project Engineer

VOLUME II
TABLE OF CONTENTS

VOLUME II
TABLE OF CONTENTS (CONTINUED)

VOLUME II
LIST OF FIGURES

1.0 SCOPE

1.1 SCOPE

This Guidebook provides procedures for the preparation of software reliability predictions and estimations for embedded and separately procured computer systems. The results of prediction and estimation are primarily intended to serve as relative indicators of reliability in connection with design decisions and in monitoring progress of a project. Caution must be used in equating predicted or estimated values of softwae reliability with operational values, as is also the case in hardware reliability prediction.

1.2 APPLICATION

The requirements and procedures established by this Guidebook may be selectively applied to any Department of Defense contract-definitized procurements, request for proposals, statements of work, and in-house Government projects for system development and production. It is not intended that all the requirements herein will need to be applied to every program or program phase. Procuring activities shall tailor the requirements of this standard to the minimum needs of each procurement and shall encourage contractors to submit cost effective tailoring recommendations.

1.3 SCOPE

Software reliability prediction and estimation techniques are described as a methodology in this guidebook for assessing a software system's ability to meet specified reliability requirements. Software reliability prediction translates software measurements taken during early life cycle phases, into a predicted reliability. Software Reliability estimation estimates, based on test phase indicators, how reliably the software will perform its required functions in its operational environment. At this time no software techniques are documented in this Guidebook to estimate the demand for maintenance and logistic support caused by software systems unreliability. When used in combination, the two techniques provide a basis for identifying areas wherein special emphasis or attention is needed, and for comparing the cost-effectiveness of various design configurations. This guidebook is intended as a companion document to MIL-STD-785B, MIL-STD-756B and MIL-HDBK 217E.

1.4 NUMBERING SYSTEM

Task sections, tasks, and methods are numbered sequentially as they are introduced into this Guidebook in accordance with the following classification system.

1.4.1 Classification of Task Sections, Tasks, and Methods

100	-	Reliability Prediction Task Section
101-199	-	Reliability Prediction Tasks
1001-1999	-	Reliability Prediction Methods
200	-	Reliability Estimation Task Section
201-299	-	Reliability Estimation Tasks
2001-2990	-	Reliability Estimation Methods

1.5 REVISIONS

1.5.1 Standard

Any general revision of this guidebook which results in a revision of sections 1, 2, 3, 4 or 5 will be indicated by a revision letter together with the date of revision.

1.5.2 Task Sections, Tasks, and Methods

Revisions are numbered consecutively indicated by a letter following the number. For example, for task 101, the first revision is 101A, the second revision is 101B. When the basic document is revised, those requirements not affected by change retain their existing date.

1.6 METHOD OF REFERENCE

The tasks and methods contained herein shall be referenced by specifying:

- This guideline number
- Task number(s)
- Method number(s)
- Other data as called for in the individual task or method

2.0 REFERENCED DOCUMENTS

2.1 ISSUES OF DOCUMENTS

The following documents of the issue in effect on date of invitation for bids or request for proposal, are referenced in this guideline for information and guidance.

STANDARDS

MIL-STD-785B	Reliability Program for Systems and Equipment Development and Production
MIL-STD-721	Definitions of Terms for Reliability and Maintainability
MIL-STD-781C	Reliability Design Qualification and Production Acceptance Tests: Exponential Distribution
MIL-STD-105	Sampling Procedures and Tabbles for Inspection by Attribute
MIL-STD-1521A	Technical Reviews and Audits for Systems, Equipment, and Computer Programs
MIL-HDBK-217E	Reliability Prediction of Electronic Equipment
MIL-STD-756B	Reliability Modeling and Prediction
MIL-STD-2167A	Defense System Software Development
MIL-STD-2168	Software Quality Evaluation (Proposed)
MIL-STD-1679	Weapon System Software Development
MIL-STD-490	Specification Practices
MIL-STD-480	Configuration, Control, Engineering Changes, Deviations, and Waivers
MIL-STD-483	Configuration Management Practices for System, Equipment, Munitions, and Computer Programs
MIL-Q-9858	Quality Program Requirements
MIL-STD-52779A	Software Quality Program Requirements

2.2 OTHER PUBLICATIONS

The following documents are potential sources of reliability data or describe techniques that may be used in conjunction with this Guidebook. Specific requirements for use of these or other data sources must be specified by the procuring activity.

RADC TR 85-37 "Specification of Software Quality Attributes", February 1985

RADC TR 85-228 "Impact of Hardware/Software on System Reliability", January 1985

RADC TR 83-176 "A Guidebook for Software Reliability Assessment", 1983

RADC TR 84-53 "Software Test Handbook", March 1984

3.0 DEFINITIONS

3.1 TERMS

Terms used in this document are as defined in Appendix A.

4.0 GENERAL REQUIREMENTS

4.1 GENERAL

Software reliability prediction and estimation shall be planned and performed in accordance with the general requirements of this guidebook and the task(s) and method(s) specified by the procuring activity.

4.1.1 The Reliability Problem

When it is proposed to design a system which includes computers to perform a complex and demanding job, it is assumed that the required investment will be justified according to the perfection by which the job is performed or by the large number of times which the system can do the job. This assumption cannot be justified when a system fails to perform upon demand or fails to perform repeatedly. Thus, the reliability of a system is critical to it cost effectiveness.

Reliability is a consideration at all levels of electronics, from materials to operating systems to application software because the components are combined in systems of ever increasing complexity and sophistication. Therefore, at any level of development and design, it is natural to find the influence of reliability engineering acting as a discipline devoting special engineering attention to the unreliability problem. Reliability engineering has been primarily concerned with the time degradation of materials, physical and electronic measurements, equipment design, processes and system analysis, and synthesis. This Guidebook extends that discipline to software reliability engineering. None of these can be isolated from the overall electronics context or software development process but must be carried on in conjunction with many other disciplines.

4.1.2 The Role of Reliability Prediction and Estimation in Software Engineering

To be of value, a prediction or estimation must be timely. However, the earlier it is needed, the more difficulties will be encountered. It is certainly true that the earlier a prediction has to be made about the unknown nature of a future event, the more difficult it is to make a meaningful prediction. As an example, it can be seen that the reliability of an electronic equipment is known with certainty after it has been used in the field and it is worn out and its failure history has been faithfully recorded. But, for purposes of doing anything about the reliability of this equipment, this knowledge has little value. Before this point, reliability cannot be known with certainty; but a great deal of knowledge about reliability can be accumulated over a short period early in the useful life. Even though the degree of certainty of knowledge is less, there is some opportunity to do something to influence the reliability of

the remaining life portion.

Similarly, considering the various stages back through installation, shipment, test, production, test design, development, procurement, etc., less and less can be known with certainty about reliability. However, what is known or predicted becomes more and more valuable as a basis for taking action. After all, there is no value in simply knowing that a certain failure will occur at some specific time in the future. The value comes in having the opportunity to do something to prevent the failure from occurring. Once this is done, the future is changed from what was predicted with certainty. Thus, prediction becomes part of a process of "designing the future".

An early prediction is made on the basis of preliminary knowledge in order to evaluate the reliability of alternative software designs, and to permit selection of an alternate that has a high likelihood of meeting the reliability objectives. The process, in order to have any meaning at all, requires predicting, acting, measuring (or gaining new knowledge), then repredicting, acting again and remeasuring continually throughout a program of development.

The two trends in the prediction art are: (1) To gain better records of class characteristics in more usable and realistic forms and (2) To develop improved techniques for applying the consequent knowledge to predictions in appropriate confidence settings. The current state-of-the-art in software reliability predictions rests at the level of development of these data and techniques. Much room remains for advancing the state-of-the-art.

4.2 Software Reliability Program

The contractor shall establish and maintain an efficient reliability program to support economical achievement of overall program objectives. To be considered efficient, a reliability program shall clearly: (1) improve operational readiness and mission success of the major end-item; (2) reduce item demand for maintenance manpower and logistic support; (3) provide essential management information; and (4) hold down its own impact on overall program cost and schedule.

4.2.1 Program Requirements

Each reliability program shall include an appropriate mix of reliability engineering and accounting tasks depending on the life cycle phase. These tasks shall be selected and tailored according to the type of item (system, subsystem or unit) and for each applicable phase of the acquisition. They shall be planned, integrated and accomplished in conjunction with other design, development and manufacturing functions. The overall acquisition program shall include the resources, schedule, management structure, and controls necessary to ensure that specified

reliability program tasks are satisfactorily accomplished. Figure 4-1 illustrates the insertion of software reliability prediction and estimation into the software development process. Note that the methodology actually spans the software life cycle including reliability specification and reliability assessment once the system is operational.

4.2.1.1 Reliability Engineering

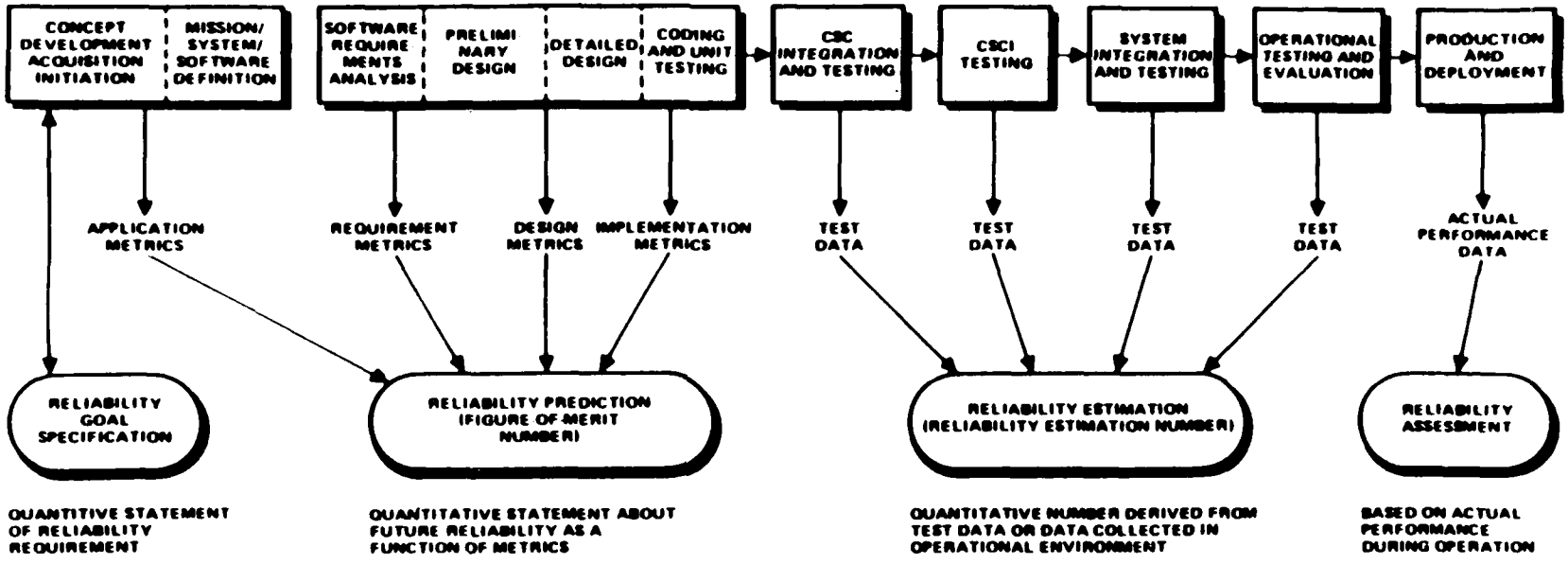
Tasks shall focus on the prevention, detection, and correction of reliability design deficiencies, unreliable units, and workmanship defects. Reliability engineering shall be an integral part of the design process, including design changes. The means by which reliability engineering contributes to the design, and the level of authority and contraints on this engineering discipline, shall be identified in the reliability program plan. An efficient reliability program shall stress early investment in reliability engineering tasks to avoid subsequent costs and schedule delays.

Figure 4-2 illustrates the software reliability prediction and estimation discipline in context of an overall approach to improving software reliability. As illustrated, the concerns with software reliability must permeate the entire software development process. In fact, these same disciplines are applicable to post deployment software support, i.e., software logistics support. The developers must approach the software development with reliability as a goal. Use of formal approaches such as MIL-STD 2167A, modern techniques and tools, provide the foundation for building reliability into the product. The testing process must also account for reliability demonstration. RADC TR 84-53, Software Test Handbook, provides a methodology for planning testing techniques and tools which aid in meeting testing objectives. The prediction and estimation techniques advocated in this document provide the oversight role. Companion documents are the proposed MIl-STD 2168, which states software QA requirements for DOD software developments; RADC TR 85-37, which establishes a methodology for quality specification and measurement; RADC TR 85-47, Impact of Hardware/Software Faults on Sytem Reliability which establishes a new modeling approach to software reliability; and RADC TR 83-176, which is a guidebook on the use of existing software reliability models.

The incorporation of this approach in software developments promises significant benefit. This general approach could be viewed as a software reliability discipline. Functions of that discipline are portrayed in Figure 4-2. The activities that comprise that discipline are indicated in Figure 4-3.

4.2.1.2 Reliability Accounting

Tasks shall focus on the provision of information essential to acquisition, development, operation, and support management, including properly defined inputs for estimates of operational



Note THE RELIABILITY FIGURE OF MERIT NUMBER AND THE RELIABILITY ESTIMATION NUMBERS MUST BE DEFINED IN THE SAME UNITS OF MEASUREMENT SO THAT THEY MAY BE MEANINGFULLY COMPARED AND RELATED TO EACH OTHER

FIGURE 4-1 FRAMEWORK FOR SOFTWARE RELIABILITY

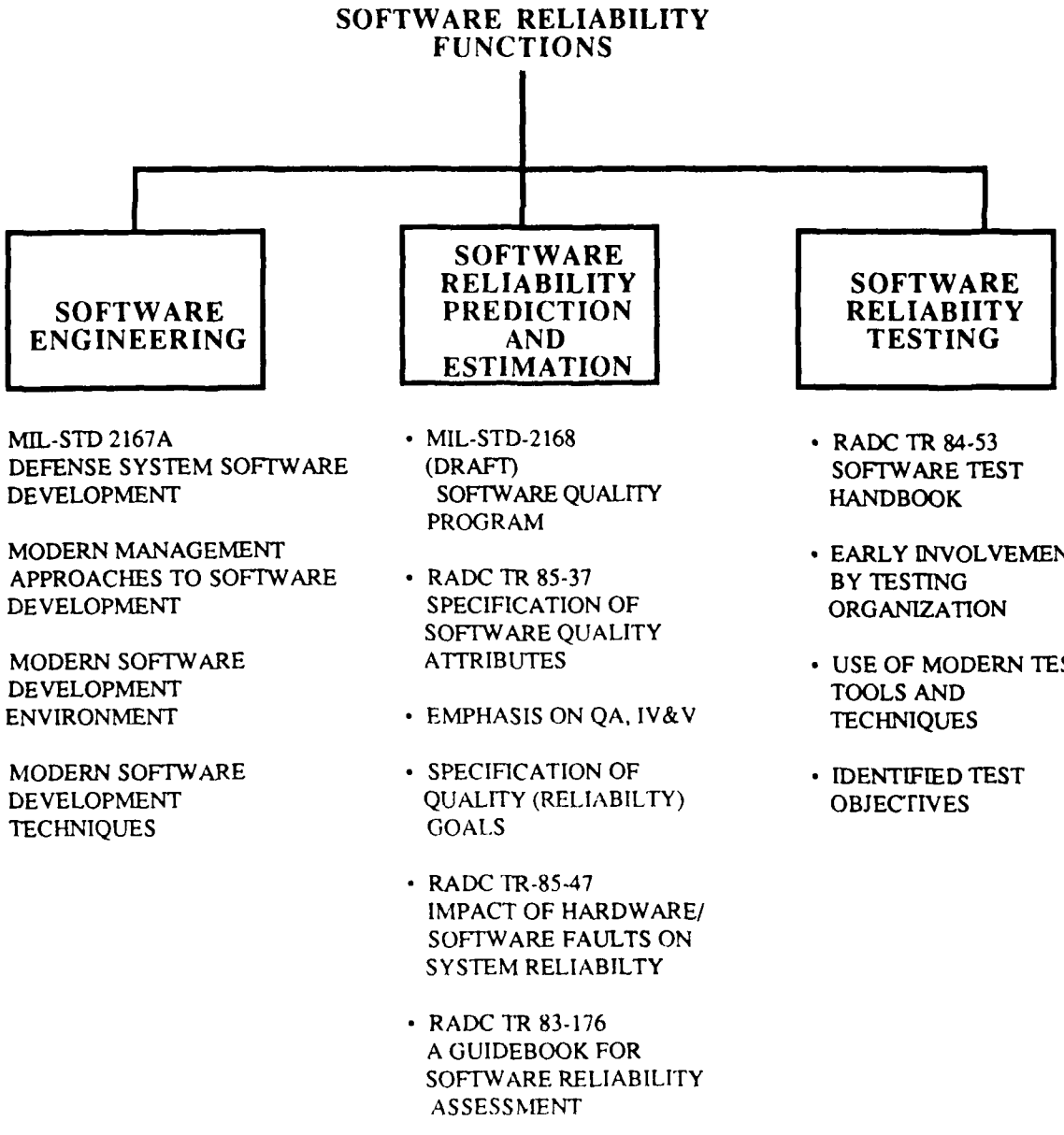


FIGURE 4-2 SOFTWARE RELIABILITY FUNCTIONS

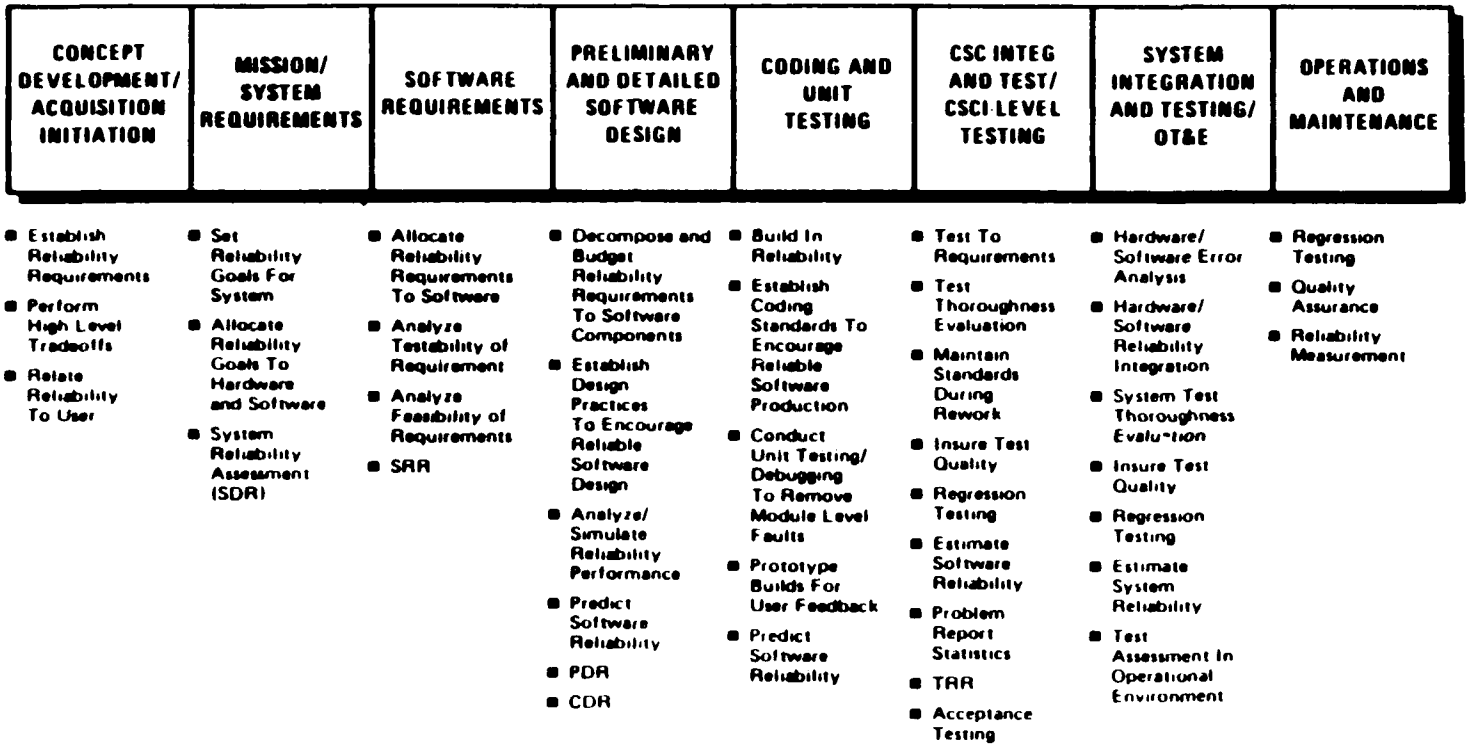


FIGURE 4-3. SOFTWARE RELIABILITY ENGINEERING MANAGEMENT

effectiveness and ownership cost. An efficient reliability program shall provide this information while ensuring that cost and schedule investment in efforts to obtain management data (such as demonstrations, qualification tests, and acceptance tests) is clearly visible and carefully controlled.

4.2.2 Reliability Program Interfaces

The contractor shall utilize reliability data and information resulting from applicable tasks in the reliability program to satisfy Post Deployment Software Support (PDSS) requirements. All reliability data and information used and provided shall be based upon, and traceable to, the outputs of the reliability program for all maintenance support and engineering activities involved in all phases of the system acquisition.

4.2.3 Quantitative Requirements

The software system reliability requirements shall be specified contractually.

4.2.3.1 Categories of Quantitative Requirements

There are three different categories of quantitative reliability requirements: (1) operational requirements for applicable software reliability parameters; (2) basic reliability requirements for software design and quality; and (3) statistical confidence/decision risk criteria for specific reliability tests. These categories must be carefully delineated, and related to each other by clearly defined audit trails, to establish clear lines of responsibility and accountability.

4.2.3.2 System Reliability Parameters

Software reliability parameters shall be defined in units of measurement directly related to operational readiness, mission success, demand for maintenance manpower, and demand for maintenance support, as applicable to the type of system. Operational requirements for each of these parameters shall include the combined effects of design, quality, operation, maintenance and repair in the operational environment. The basic measurement used in this guidebook for software reliability is failure rate. Definitions are provided in Appendix A.

4.2.4 Limitations of Reliability Predictions

The art of predicting the reliability of software has practical limitations such as those depending on data gathering and technique complexity. Considerable effort is required to generate sufficient data to report a statistically valid reliability figure for a class of software. Casual data gathering accumulates data more slowly than the advance of technology; consequently, a valid level of data is never attained. In the case of software, the number of people participating in data

gathering all over the industry is rather large with consequent varying methods and conditions which prevent exact coordination and correlation. Also operational software reliability data is difficult to examine due to the lack of suitable data being acquired. Thus, it can be seen that derivation of failure rates (being mean values) is empirically difficult and obtaining valid confidence values is practically precluded because of lack of correlation.

The use of failure rate data, obtained from field use of past systems, is applicable on future concepts depending on the degree of similarity existing both in the software design and in the anticipated environments. Data obtained on a system used in one environment may not be applicable to use in a different environment, especially if the new environment substantially exceeds the design capabilities. Other variants that can affect the stated failure rate of a given system are: different uses, different operators, different maintenance practices, different measurement techniques or definitions of failure. When considering the comparison between similar but unlike systems, the possible variations are obviously even greater.

Thus, a fundamental limitation on reliability prediction is the ability to accumulate data of known validity for the new applications. Another fundamental limitation is the complexity of prediction techniques. Very simple techniques omit a great deal of distinguishing detail and the prediction suffers inaccuracy. More detailed techniques can become so bogged down in detail that the prediction becomes costly and may actually lag the principal development effort.

This Guidebook includes two methods: reliability prediction and reliability estimation. These methods vary in degree of information needed and timing of their application. References to other or complementary methods are provided.

The content of this Guidebook has not been approved by the Military Services and has not been coordinated with appropriate segments of industry. It provides an initial attempt to document a methodology that would provide a common basis for reliability predictions during acquisition programs for military systems. It also establishes a common basis for comparing and evaluating reliability predictions of related or competitive designs. The failure rates and their associated adjustment factors presented herein are based upon evaluation and analysis of the best available data at the time of issue.

4.3 IMPLEMENTATION

Reliability prediction shall be initiated early in the definition stage to aid in the evaluation of the system architecture and design and to provide a basis for system reliability allocation (apportionment) and establishing corrective action priorities.

Reliability estimation shall be initiated early in the test phases utilizing the observed failure rate during testing as a basis to estimate how the software will behave in an operational environment. Reliability predications and estimations shall be updated when there is significant change in the system design, availability of design details, environmental requirements, stress data, failure rate data, or service use profile. A planned schedule for updates shall be specified by the procuring activity.

4.4 GROUND RULES AND ASSUMPTIONS

The Government Program Office or contractor shall develop ground rules and analysis assumptions. The ground rules shall identify the reliability prediction and estimation approach in terms of this Guidebook, the lowest indenture level to be analyzed, and include a definition of mission success in terms of performance criteria and allowable limits. The SPO or contractor shall develop general statements of item mission success in terms of performance and allowable limits for each specified output. Ground rules and analysis assumptions are not inflexible and may be added, modified, or deleted if requirements change. Ground rules and analysis assumptions shall be documented and included in the reliability prediction and estimation report.

4.5 INDENTURE LEVEL

The indenture level applies to the software or functional level at which the software configuration is defined. Unless other wise specified, the contractor shall establish the lowest indenture level of analysis using the following guidelines:

- The level specified for the prediction measurement to ensure consistency and allow cross referencing.
- The specified or intended maintenance level for the software.

The methodology described in this guidebook supports reliability prediction and estimation at the system, CSCI, CSC, and unit levels.

4.6 CODING SYSTEM

For consistent identification of system functions and software elements, the contractor shall adhere to a coding system based upon the software breakdown structure, work unit code numbering system of MIL-STD-780, or other similar uniform numbering system. The coding system shall be consistent with the functional block diagram numbering system to provide complete visibility of each modeled element and its relationship to the item.

4.7 MISSION RELIABILITY DEFINITION

System reliability for mission is assumed to be represented by a series arrangement of hardware, software, and possibly other components as shown in figure 4-4. The mathematical formulation for the system mission reliability is therefore

$$R=R_H*R_S*R_X$$

Hardware-software interactions, such as software failures induced by hardware anomalies, or failures of hardware reconfiguration caused by software faults, must be included in the R_X term. Other components that may have to be added to the series model include the personnel subsystem and support equipment (power, airconditioning, etc.). Only the prediction or estimation of the R_S component is covered by this Guidebook.

If the reliability of individual components is high, eg. at least 0.95, a good approximation of the system reliability can be obtained by

$$F=F_H+F_S+F_X$$

where all F terms are mission failure probabilities (R=1-F). The software mission failure probability is the product of the software failure rate and the mission duration, expressed in identical units of time.

Where mission phases differ in hardware or software utilization or environment, a separate reliability model is required for each phase, and the total mission reliability is the series combination (product) of the individual mission phases. Differences in software utilization are presented if (a) functionally distinct software is utilized, such as automatic approach and landing software in an aircraft flight control system, or (b) there is a substantial difference in the mix of software functions. Differences in the software environment are present if there are substantial changes in the computer workload.

4.8 COORDINATION OF EFFORT

Reliability and other organizational elements shall make coincident use of the reliability predictions and estimations. Considerations shall be given to the requirements to perform and use the reliability predictions and estimations in support of a reliability program in accordance with MIL-STD-785B, maintainability program in accordance with MIL-STD-470, safety program in accordance with MIL-STD-882, survivability and vulnerability program in accordance with MIL-STD-2072, logistics support analysis in accordance with MIL-STD-1388, maintenance plan analysis (MPS) in accordance with MIL-STD-2080, fault diagrams analysis in general accordance with MIL-STD-1591, and other contractual provisions.

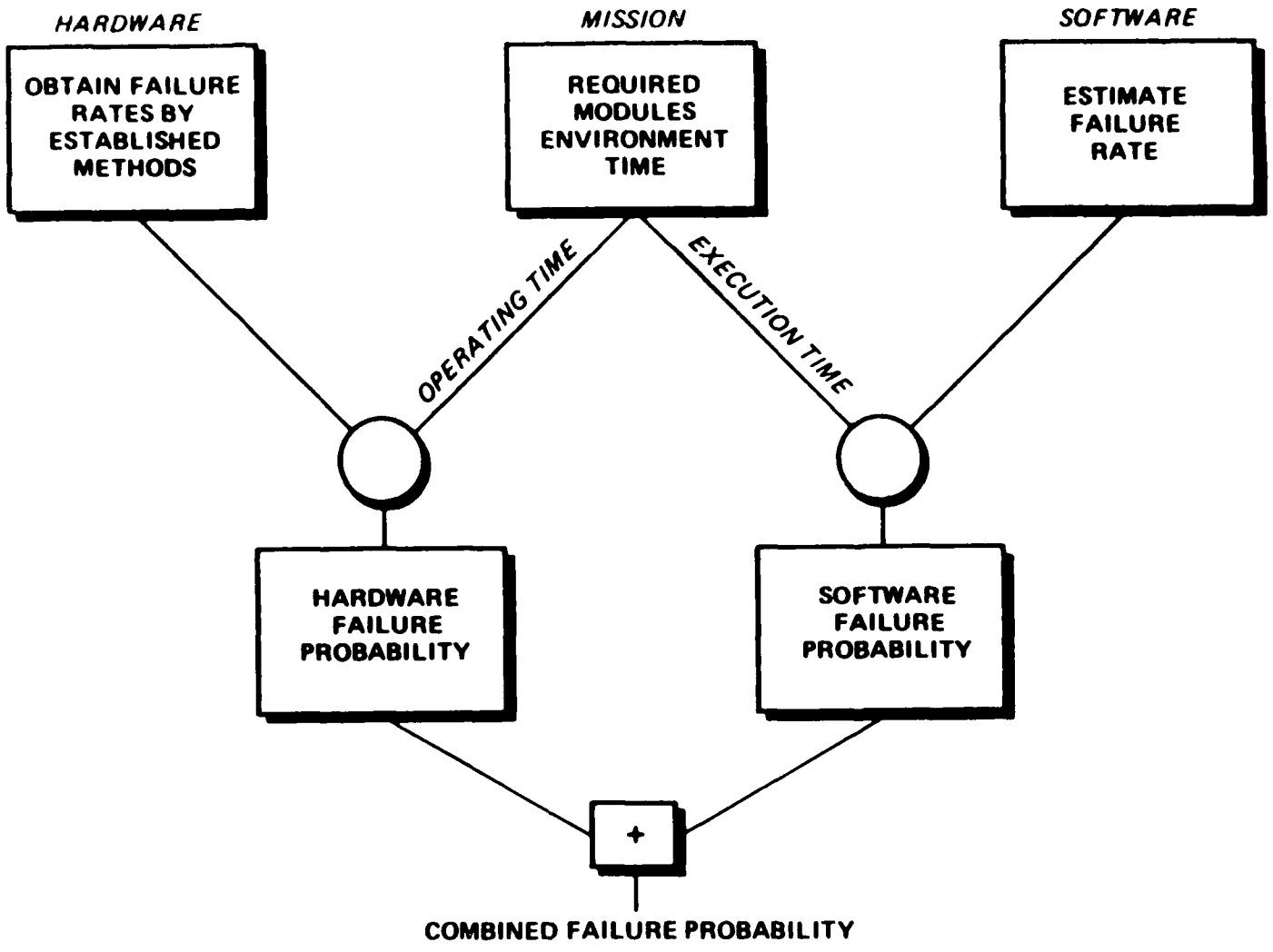


FIGURE 4-4 RELATIONSHIP BETWEEN HARDWARE AND SOFTWARE RELIABILITY

4.9 GENERAL PROCEDURE

The steps set forth below define the general procedure for developing a software reliability model and performing a reliability prediction and estimation. Specific tasks are contained in the Task Sections in Section 5. Figure 4-5 provides a road map for use of the procedures and tasks. Effort to develop the information for the steps below shall be closely coordinated with related program activities (such as design engineering, system engineering, maintainability, and logistics) to minimize duplications and to assure consistency and correctness.

- Define the software component level for prediction (See paragraph 4.9.2).
- Identify Life Cycle and Prediction and Estimation Milestones (See paragraph 4.9.3).
- Identify Data Collection Procedures (See Appendix B).
- Obtain or Develop System Architecture Diagram to Appropriate Component Level (requires allocation of software component to hardware components) (See Reliability Prediction Task Section 100).
- Define Software Components (See Task Section 100).
- Define Reliability Model (See Task Section 100).
- Implement data collection procedures (See Appendix C and D).
- Proceed through Prediction Procedures (See individual Reliability Prediction Tasks 101 through 104).
- Preceed through Estimation Procedures (See individual Reliability Estimation Tasks - 201 through 202).

4.9.1 Comparison with Hardware Reliability Prediction

Reliability prediction for hardware is an established technique, and it is therefore useful to compare the proposed software reliability procedures with those in use in the hardware field. The governing document for hardware reliability prediction for DoD applications is MIL-STD-756B "Reliability Modeling and Prediction", and MIL-STD-785B, "Reliability Program for Systems and Equipment Development and Production". The essential steps for reliability prediction identified in MIL-STD-756B have parallel equivalent procedures for software with one exception. That exception is the absence of software equivalents for step e. Hardware components consist of separate parts, each of which may be used in many other applications, such as a 1A 250V diode or a 16k dynamic RAM chip. Failure rates can be established for these

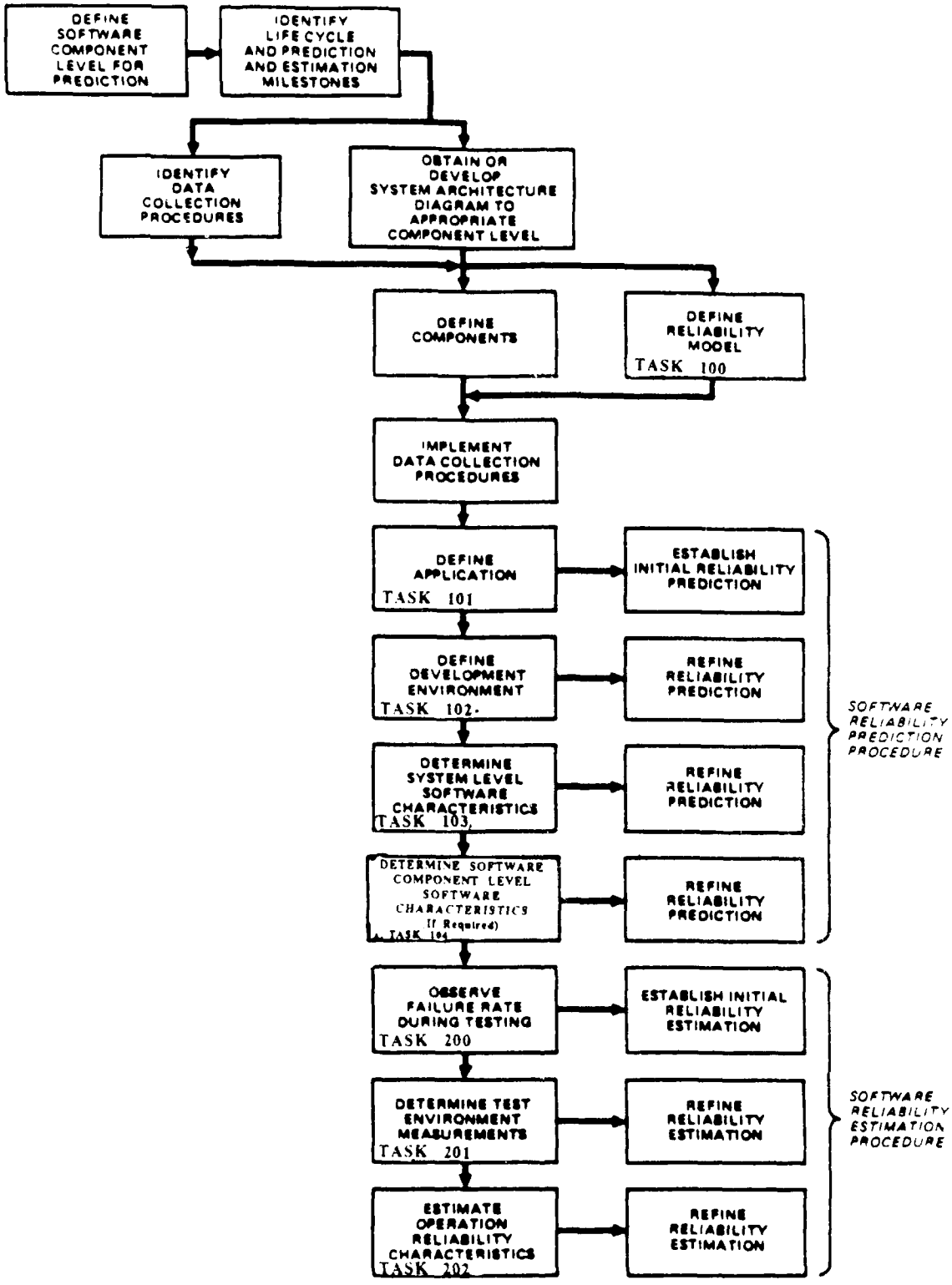


FIGURE 4-5 SOFTWARE RELIABILITY PREDICTION AND ESTIMATION PROCEDURES

parts either from test or from analysis of field data. The procedures of MIL-STD-756B assume that the reliability of a component is the product of the reliability of its (series-connected) parts. The software analog to this would be to test individual assignment, branching, and I/O statements and to declare the reliability of a procedure to be the product of the reliability of its individual statements. This analog is faulty because: (a) statements cannot be meaningfully tested in isolation and (b) many software failures arise not from faults in a single statement, but rather from interactions between multiple statements (or from interactions between hardware and software). As reuseable software gains wider acceptance, the assignment of a reliability index (equivalent to parts failure rate) to standard procedures may become practicable but this is still in the future.

The application of the other steps to software reliability prediction is discussed below.

The following paragraphs describe the application to software reliability prediction of those steps of the MIL-STD-756B procedure that were found to be suitable. the steps are numbered here, but the lower case alphabetical designation from MIL-STD-756B is indicated in parentheses for ease of reference. Only asterisked steps are required for the prediction of fault density. These steps have been taken into account in Figure 4-5.

1. (a)* Define the software components to be covered by the prediction. This includes an unambiguous identification of the component, a statement of the performance requirements and the hardware environment, and a listing of inputs and outputs by type and range. This information may be available initially only at a high level of abstraction but should be decomposed for permitting tracing predictions during successive stages of development, and comparing predictions with estimates and measurements during later periods.

2. (b) Define the life cycle stages to be covered by the prediction and the extent of use during each stage. It is recognized that the failure rate of software is a function of the life cycle stage. Particularly, there are significant differences in the failure rate between test and operations, and between initial operation and mature operation. Therefore, the life cycle stage(s) for which the prediction is to be made must be identified. The probability of fault removal depends on the extent to which the software is exercised. Therefore the use (in CPU-hours) between the time the prediction is made and the target for the prediction must be known.

3. (c) Define the execution dependencies within the software component. This will in general require review

of a top level flow chart or block diagram of the software component in order to identify units (a unit in this context is a software element at or above the module level) that are executed:

- Routinely -- during every invocation of the software component, or once during each defined cycle for iterative programs (e.g., closed loop control);

- Irregulary -- segments that deal with non-routine events within the program, including exceptions to conditions postulated within the program (but not exception states of computer or operating system);

- Conditionally -- segments that are executed only if some other (non-routine) segment had been invoked (examples are message logging or creations of new files);

- For exception handling -- response of the program to exception states identified by the computer or operating system;

- On demand -- segments accessed only by specific operator actions such as initialization, data base cleanup, or rehosting.

  Discussion of a technique to represent execution dependencies is found in RADC TR 85-47.

4. Since both the probability of execution and the accumulated execution time will differ between these classifications, separate reliability predictions will usually be required.

5. (d) Define mathematical models for the software components. The mathematical models will represent:

   - The predicted fault density of each segment as derived in the next section;

   - The execution time of each segment prior to the prediction interval -- to determine the expected fault removal; and

   - The execution time of each segment during the prediction interval -- to determine the failure probability.

     Where the prediction interval covers more than one life cycle phase (such as test and operation) a separate mathematical model will be required for each phase.

6. (e) Define and describe the parts of the item. Use application area factor. As discussed in the preceding section, a major divergence of software from hardware reliability prediction practices is due to lack of an equivalent to the hardware part. However, software reliability prediction is still based on concepts of quantity, the average number (fraction) of faults per line of code. The number of faults in a software component is thus assumed to be proportional to the number of lines of code. Although we cannot, at the present state of knowledge, identify one computer program as being made up of high failure rate parts and another one of low failure rate parts, there is evidence that high and low failure fault densities are associated with certain application areas. The application area factor captures this experience as a basic predictor of the fault density.

7. (f) Define the operational environment. The operational environment determines the rate at which the faults inherent in the software will be transformed to failures. Operational environment in this sense means the environment in which the software will be operating during the interval for which the reliability prediction is to be made. It can apply to test, operation in a prototype environment, or a full scale operational environment. The most important characteristics of the operational environment which affect the reliability are:

   - Computer performance (throughput),
   - Variability of Data and Control States, and
   - Workload.

   The contribution of each of these to the reliability estimation is discussed in Section 5.

8. (g)* Account for software development environment and software implementation. Differences in the software development environment and in the software implementation affect the fault density in a manner similar to that in which stress levels affect the failure probability of parts.

9. (h) Define the failure distribution in execution time. Software fails only when it is being executed. Therefore, the natural normalization factor for software failures is execution time. The software failure rate bsaed on Computer Operation hour is analogous to the hardware failure rate ("lambda") per hour (implying operating hour of the component).

10. (i)* Compute the Reliability. The algorithms for predicting fault density are discussed in Section 5 as well as the conversion of fault density into failure rate. The estimation of reliability based on testing experience is also described in Section 5.

4.9.2 Software Component Level

The initial step in following the prediction and estimation procedures is the determination of the level at which the software reliability will be modeled. The levels of software are defined by MIL-STD 2167A as System, Computer Software Configuration Item (CSCI), Computer System Component (CSC), and Unit. The Reliability prediction and estimation procedures on this Guidebook can be used at any of these levels.

The following procedure is recommended.

- During early phases of development (Concept Development, Mission System/Software Definition, Software Requirements) model at software system level.
- During design phases, model at CSCI level.
- During coding model at the CSC level. For critical software the contracting agency may direct modeling at a lower level (such as unit). Support software or commercial off-the-shelf programs should be modeled at the CSCI level or system level.
- During testing, model at CSCI level or, if directed, a lower level.

4.9.3 Identify Life Cycle

The software life cycle according to MIL-STD 2167A is illustrated in Figure 4-1. Applicable points during this life cycle when a reliability prediction or estimation is recommended are:

- During Concept Development to support Feasibility Studies.
- During Mission/System/Software Definition to support high level architectural studies/tradeoff studies and to establish development goals/specifications. Results should be reported formally at SDR.
- During proposal preparation by contractors for evaluation purposes.
- During Software Requirements Analysis to support feasibility analyses. Results should be reported formally at SRR.
- During Preliminary Design to support software architecture decision and allocation. Results should be reported

formally at PDR.

- During Detailed Design to support detailed design decisions/tradeoff studies/algorithm development. Results should be reported formally at CDR.

- During coding and unit testing to support developer's decision to release software to formal testing. Results can be reported through QA audit reports or problem reporting process.

- During testing phases to support test and evaluation process and acceptance. Results can be reported at the end of each phase of testing or periodically during testing. Results of any acceptance testing should be formally reported.

- During OT&E as formal evaluation process.

- During post deployment support as an assessment of actual reliability achieved and to support a Reliability Improvement Program.

4.10 RELIABILITY PROGRAM PLAN AND RELIABILITY MODELING AND PREDICTION REPORT

4.10.1 Software Reliability Program Plan

A Reliability Program Plan shall be prepared and include, but not limited to the following:

a. Recognition of the Reliability Program within the development organization responsible for the development.

b. Description of the software reliability requirements established for the system and their relationship with the system reliability requirements.

c. Description of how the Software Reliability Program will be conducted to meet the software reliability requirements.

d. Establishment of responsible personnel for the conduct of the Reliability Program with appropriate authority.

e. A description of the relationship of the Reliability Program with appropriate authority.

f. A schedule (see previous paragraph 4.9) of the reliability prediction and estimation activities (milestones).

g. Identification of data collection requirements and procedures to support the reliability prediction and estimation activities.

h. Description of the reliability prediction and estimation procedures to be used.

i. Identification of potential or known reliability problems.

j. Procedures for recording the status of actions to resolve the problems identified.

4.10.2 Software Reliability Modeling and Prediction Report

The reliability models and reliability predictions and estimations shall be documented in a report that identifies the level of analysis, summarizes the results, documents the data sources and techniques used in performing the analysis, and includes the component definition narrative, resultant analysis data, worksheets, ground rules and assumptions. Interim reports shall be available at each design review to provide comparisons of alternative designs and to highlight high failure rate elements of the design, and proposed design corrections or improvements. The final report shall reflect the final design and provide identification of potentially high failure rate software elements and of software elements that are especially critical to mission success. When submitting a report applicable for an Exploratory/Advanced Development Model, a simplified reliability modeling and prediction report is appropriate.

4.10.2.1 Summary

The report shall contain a summary which provides the contractor's conclusions and recommendations based upon the analysis. Contractor interpretation and comments concerning the analysis and the recommended actions for the elimination or reduction of failure risks shall be included. A design evaluation summary of major problems detected during the analysis shall be provided in the final report. A list of software or functional elements of the system omitted from the reliability models and reliability predictions shall be included with rationale for each element's exclusion.

4.10.2.2 Reliability Critical Element Lists

Reliability critical software components of the system extracted from the reliability modeling and reliability prediction effort shall be listed and included in the summary. Reliability critical software components include high failure rate components (experienced during testing), real-time processing components, and those components performing mission critical functions.

4.10.2.3 Prediction and Estimation Methods

The data collected and results of the methods and procedures described in Section 5 of this guideline should be provided as appendicies to this report to substantiate the summary conclusions and the identified critical elements.

4.11 SOFTWARE RELIABILITY PROGRAM TASKS

In addition to developing the Software Reliability Program Plan, using the procedures in Task Sections 100 and 200 of this guidebook to do reliabilty predictions and estimations, and documenting the results in a Reliability Prediction Report, other tasks to be performed with the Software Reliability Program are:

MONITOR/CONTROL OF SUBCONTRACTORS AND SUPPLIERS

The contractor shall assure that software components obtained from subcontractors or suppliers meet reliability requirements. The contractor shall, as appropriate:

a. Incorporate quantitative software reliability requirements in subcontracted software specifications.

b. Assure that subcontractors have a Reliability Program that is compatible with the overall program and includes provisions to review and evaluate the software to be delivered.

c. Attend and participate in subcontractors design reviews.

d. Review subcontractors predictions and estimations for accuracy and correctness of approach.

e. Review subcontractor's test plans, procedures, and reports.

f. Require delivery of appropriate data collected in accordance with the Reliability Program.

g. Assure the subcontractors have and are complying with corrective action reporting procedures and follow-up corrective actions.

h. Monitor reliability demonstrations tests.

A reference doucment is MIL-STD 52779A.

PROGRAM REVIEWS

The Reliability Program shall be planned and scheduled to permit the contractor and the Government to review program status.

Formal review and assessment of contract reliability requirements shall be conducted at major program points, identified as system program reviews, as specified by the contract. As the program develops, reliability progress shall also be assessed by the use of additional reliability program reviews as necessary. The contractor shall schedule reviews as appropriate with his subcontrators and suppliers and insure that the Government is informed in advance of each review.

The reviews shall be identified and discuss all pertinent aspects of the reliabilty program such as the following, when applicable:

a. At the Software Requirements Review

   (1) Identify reliability requirements in terms of fault density and failure rate (see Table TS101-1).

   (2) Establish allocation of software reliability requirements to software components (CSCI).

b. At the Preliminary Design Review (PDR):

   (1) Update reliability status including:

      a. Reliability modeling
      b. Reliability apportionment
      c. Reliability predictions
      d. Failure Modes, Effects and Criticality Analysis (FMECA)
      e. Reliability content of specification
      f. Design guideline criteria
      g. Other tasks as identified

   (2) Other problems affecting reliability

   (3) Reliability critical items program.

c. At the Critical Design Review (CDR):

   (1) Reliability content of specifications

   (2) Reliability prediction and analyses

   (3) Reliability critical items program

   (4) Other problems affecting reliability

   (5) FMECA

d. At Interim Reliability Program Reviews:

   (1) Discussion of those items reviewed at PDRs and CDRs

   (2) Results of failure analyses

(3) Test schedule: start dates and completion dates

(4) Component design, reliability, and schedule problems

(5) Status of assigned action items

(6) Contractor assessment of reliability task effectiveness

(7) Other topics and issues as deemed appropriate by the contractor and the Government.

e. At the Test Readiness Review:

(1) Reliability analyses status, primarily prediction

(2) Test schedule

(3) Test profile

(4) Test plan including failure definition

(5) Test report format

(6) FRACAS implementation

A reference document is MIL-STD 1521A.

FAILURE REPORTING, ANALYSIS, AND CORRECTIVE ACTION SYSTEM (FRACAS)

The contractor shall have a closed loop system that collects, analyzes, and records failures that occur for specified levels of the software prior to acceptance by the procuring activity. The contractor's existing data collection, analysis and corrective action system shall be utilized, with modification only as necessary to meet the requirements specified by the Government.

Procedures for initiating failure reports, the analysis of failures, feedback of corrective action into the design, manufacturing and test processes shall be identified. The analysis of failures shall establish and categorize the cause of failure.

The closed loop system shall include provisions to assure that effective corrective actions are taken on a timely basis by a follow-up audit that reviews all open failure reports, failure analyses, and corrective action suspense dates, and the reporting of delinquencies to management. The failure cause for each failure shall be clearly stated.

When applicable, the method of establishing and recording operating time shall be clearly defined.

The contractor's closed loop failure reporting system data shall be transcribed to Government's forms only if specifically required by the procuring activity. Appendicies B, C, and D provide appropriate forms. A reference document is MIL-STD 785B.

FAILURE REVIEW BOARD (FRB)

The FRB shall review functional/performance failure data from appropriate inspections and testing including subcontractor qualification, reliability, and acceptance test failures. All failure occurrence information shall be available to the FRB. Data including a description of test conditions at time of failure, symptoms of failure, failure isolation procedures, and known or suspected causes of failure shall be examined by the FRB. Open FRB identified items shall be followed up until failure mechanisms have been satisfactorily identified and corrective action initiated. The FRB shall also maintain and disseminate the status of corrective action implementation and effectiveness. Minutes of FRB activity shall be recorded and kept on file for examination by the procuring activity during the term of the contract. Contractor FRB members shall include anappropriate representitive to the FRB as an observer. If the contractor can identify and utilize an already existing and operating function for this task, then he shall describe in his proposal how that function, e.g., a Configuration Control Board (CCB), will be employed to meet the procuring activity requirements. This task shall be coordinated with Quality Assurance organizations to insure there is no duplication of effort. A reference document is MIL-STD 785B.

CRITICAL RELIABILTY COMPONENT IDENTIFICATION

Based on the Software Reliability Program, the predictions and estimations, and other analyses and tests, the contractor shall identify these software components which potentially have high risk to system reliability. Techniques such as Failure Modes, Effects and Criticality Analysis (FMECA), Sneak Circuit Analysis (SCA), Design and Code Inspections, Walk throughs, etc. are recommended to assist in this identification process. A reference doucment is MIL-STD 785B.

TEST PROGRAM

The Reliability Program shall be closely coordinated with the Test Program. The Test Program shall include a Reliability Qualification Test to demonstrate achievement of the reliability requirements. The Test Program shall be specified by reference to appropriate Military Standards. Reference documents are MIL-STD 781C and MIL-STD 2167A.

TASK SECTION 100

SOFTWARE RELIABILITY PREDICTION

100.1 PURPOSE

The purpose of task 100 is to describe the general procedures for predicting software reliabilty, in terms of FD based on character of the application Development Environment and Software Implementation.

100.2 DOCUMENTS REFERENCED IN TASK SECTION 100

MIL-STD 2167A

MIL-STD 2168 (proposed)

RADC TR 85-37

RADC TR 85-47

MIL-STD 756B

MIL HDBK 217D

MIL-STD 785B

100.3 PROCEDURES

Make Reliability Prediction

Use the measurements in Tasks 101, 102, 103 and 104 to predict reliability as follows:

- Project Initiation: Use metric A as prediction: Rp = A.
- Requirements and Design Phases: Use metrics A, D, S1: RP = A*D*S1.
- Coding Phase: Use metrics A, D, S1, S2: RP = A*D*S1*S2

100.3.1 SYSTEM ARCHITECTURE

A system architecture diagram should be obtained or developed. This diagram should show a high level allocation of software components (typically at the CSCI level) to hardware components. If available, control flow and or data flow diagrams prepared by the design team are valuable for preparation of the reliability model.

100.3.2 DEFINITION OF COMPONENTS

Each software component to be modeled should be identified and defined. This information is typically available in a system/ subsystem specification. See Section 4.9.2 of for a description of suitable component levels.

100.3.3 RELIABILITY MODEL

Based on the system architecture diagram, the software components allocated to hardware components can be identified. This allocation should be overlayed on the hardware reliability block diagram. The reliability block diagram shows interdependencies among all elements or functional groups of the system. The purpose of the reliability block diagram is to show by concise visual shorthand the various series - parallel block combinations (paths) that result in successful mission performance. A complete understanding of the system's mission definition and service use profile (operational concept) is required to produce the reliability diagram.

At this point, two approaches can be taken. The first is to utilize the prediction techniques described in Tasks 101 through 104 to calculate a Reliability Figure of Merit (RP)for each component identified in the block diagram. This is typically done at a CSCI level. The second approach is to model at a lower level the software processing within each software component.

100.3.3.1 Reliability Model 1

For each software component or component grouping on the block diagram, follow tasks 101 through 104. These tasks provide the procedures for calculating a predictive Reliability Figure of Merit (RP) according to the following equation:

$$RP = A * D * S$$

where RP is the predicted fault density, A the application type metric, D the software development environment metric, and S the software characteristic metric. A is expressed in (fractional) faults per line of code, and examples of actual values are presented in Task 101. D and S are modification factors, and each of these can have a value that is less than one (1) if the environment or implementation tends to reduce the fault density, or a value of greater than one if it tends to increase fault density. These factors are equivalent to pi factors in MIL HDBK 217E. The Application Area metric represents an average or baseline fault density which can be used as a starting point for the prediction. The Tasks 101 through 104 are preliminary procedures for prediction. The tables, coefficients, and algorithms will be updated as a result of data collection and statistical analyses being performed on more software systems. Refer to Data Collection Procedures (Appendix B) and Worksheets

(Appendix C and D) to comply with these prediction procedures. This is a generic procedure and should be applicable to all software components.

100.3.3.2 Reliability Model 2

For specified software components, a detailed model based on a functional flow analysis can be developed. A functional decomposition of the software component is required as well as a mission thread analysis. For each subcomponent as defined by the procuring authority, the procedures described in 100.3.3.1 can be used to devise an RP. The flow between these subcomponents with individual reliability numbers can be modeled as a Markov Process. RADC TR 85-47 describes this modeling approach.

100.4 SOFTWARE RELIABILITY PREDICTION

The results of using Reliability Model 1 or 2 is a prediction of software reliability for each block in the system/hardware block diagram. A description of the format and documentation required for a block diagram is in MIL-STD 756B, Task Section 100. The software reliability prediction numbers should be entered on the block diagram and incorporated into the mathematical model of that diagram. The use of these procedures and assumptions made should be documented under paragraph 2.3.8.1, Software Reliability Assumptions, in that task section.

When using Model 1, the predicted software reliablity figure of merit is a fault density as described above. When using Model 2, the predicted software reliability figure of merit is a probability that the software will not cause failure of a mission for a specified time under specified conditions. The probability of failure for a specified period of time is given by the failure rate, the expected (average) number of failures per unit time, usually taken as a computer- or CPU-hour. Because the failure rate has a direct correspondence to the definition of software reliability, it is selected as the primary unit of measure for software reliability.

The fault density, predicted by Model 1 is used as an early indicator of software reliability based on the facts that: (1) the number of problems being identified and an estimate of size are relatively easy to determine during the early phases of a development and (2) most historical data available for software systems support the calculation of a fault density, but not failure rate. Fault density is the number of faults detected (or expected to be detected) in a program divided by the number of executable lines. Fault density was found to range from 0.005 to 0.02 in high quality software, in early research on software reliability. The prediction of fault density does not require knowledge of the execution environment, and thus it is suitable for the early stages of software development. As information about the intended execution environment becomes available, the predicted fault density can be translated into a predicted

failure rate.

The fault density cannot be used directly in the system block model. Instead it can be used as an indicator for unreliable components or critical reliability components. The fault density derived by the prediction methods can be compared to Table TS101-1 which contains industry averages or with the specified fault density requirement, if stated in the RFP. Actions can then be taken in the early phases of development to remedy pinpointed unreliable components through redesign, reimplementation or emphasis and rework during test.

A transformation mechanism between fault density and failure rate is based on the following. A faulty statement will not result in a failure under any circumstances until it is executed, i.e., until it affects either the memory content or the control state of a computer. Given that a fault exists, the probability of initiating a failure is dependent on three characteristics of the execution environment:

- Computer performance (throughput),
- Variability of data and control states, and
- Workload.

These characteristics affect both test and operation and the metrics applied to them are discussed under separate headings later.

The following three approaches can be used for the transformation:

- Using established empirical values, such as are included in Table TS101-1.
- Developing a theoretically based transformation function.
- Using in-house data to derive an empirical relationship.

As a baseline for the transformations discussed here, Table TS101-1 provides currently available data. Using the Average row, a transformation ratio of .1/.0094 = 10.6, operational failure rate to fault density, is a baseline transformation ratio. Examination of Table TS101-1 shows that for individual application categories contributing to that average, the transformation ratio ranges from 1.2 to 23. Table TS100-1 is provided as currently available transformation ratios for the individual application areas.

The second approach requires the following deviation and data collection. Practically all software failure rate models postulate a direct functional relationship between the fault content of a program and its failure rate. In the simplest case,

TABLE TS100-1
TRANSFORMATION FOR
FAULT DENSITY TO FAILURE RATE

APPLICATION TYPE	TRANSFORMATION RATIO
AIRBORNE	6.2
STRATEGIC	1.2
TACTICAL	13.8
PROCESS CONTROL	3.8
PRODUCTION CENTER	23
DEVELOPMENTAL	NOT AVAILABLE
AVERAGE	10.6

the functional relation is a constant, e.g., the failure (hazard) rate is proportional to the expected value of the number of faults remaining.

These relations permit the estimation of fault content, given the failure rate, or vice versa. Two cases are the estimation of the number of faults removed in a give time interval (expressed in terms of execution time). For the first of these we use:

$$L - L_O * \exp(-Qt/No)$$

where L is the failure rate at time t, L_O is the initial failure rate, Qt is a factor that is considered constant in a given environment, and No is the initial fault content. Given the program size, the fault content can be converted to fault density.

The number of faults removed during a time interval can be obtained from:

$$n = Q1 * (L1 - L2)$$

where n is the fault decrement, Q1 a constant in a given environment and L1 and L2 are the failure rates at the beginning and end of the period over which the fault removal is estimated.

In spite of the mathematical simplicity of these formulas, considerable effort is usually required to find values for the constants Qt and Q1 that are applicable to generic environments.

The third approach requires that failure rate data be collected during operation of the software and compared with the fault density recorded during the development. This is possible if parts of the system are implemented prior to other parts, i.e., an incremental development, and those parts that are implemented early are put through an IOT&E phase of testing. Another situation where data may be available is in an environment where a new system or a new generation of an old system is being developed and existing fault density and failure rate data has been collected on the existing system and can be compared with the new development. Data Collection Procedures 5, 6, and 7 in Appendix B can be used to calculate fault density.

If one of the empirical approaches (first and third approach) is used, the computer throughput must be taken into account if the baseline is derived from a different computer than the target for the intended application. Computer performance determines the frequency with which statements are executed. All other things being equal, a program continuously executing on a fast computer will experience a higher failure rate than the same program executing on a slower computer.

Failure rates expressed in computer-hours (also referred to as wall-clock-hours) or CPU-hours are the most useful reliability

metrics in a given environment, but it must be recognized that the failure exposure of a program is dependent on the number of executions rather than on passage of time. Thus, if one pass through a program with a given data set takes 1 second on computer A and 0.1 second on computer B, then the failure exposure per unit time imparted by the latter is ten times that of the former. Other things being equal, one expects the failure rate (expressed in common time units) in B to be ten times that of running the program in A.

The customary measure of computer performance is the instruction processing rate expressed as MIPS (Million Instructions Per Second). Although this relates to the native instruction set of each computer, and is therefore not strictly comparable across computer types, it can form a working basis for most of the transformations required for reliability prediction.

A faulty program executing on a computer, even on a very fast computer, will not experience software failures if it constantly operates on a data set that has already been run correctly. On the other hand, introducing deliberate variability into the input data, as in a test environment, will accelerate the occurrence of failures. Thus, metrics for capturing the variability of the environment are an important component of the transformation procedures.

The workload of the computer system affects the software failure rate even if the execution frequency of a given program is held constant (e.g., in a multi-tasking environment where the workload is a composite of several programs). It has been found that at very high workloads, the failure rate can increase by more than an order of magnitude over the baseline (low workload) rate. Suitable metrics are discussed in later sections.

The primary use of the transformation mechanism is to permit reliability prediction using fault density level to be translated to failure rate prediction.

100.5 DETAIL TO BE SPECIFIED BY THE PROCURING AUTHORITY.

a. Requirement of tasks 101 through 104.

b. Statement of reliability requirements.

c. Define the software component level for prediction (different levels may be specified for each life cyle phase).

d. Define life cycle phases to be covered and prediction milestones.

e. Identify data collection procedures (see Appendix B of this Guidebook).

d. Identify fault density/failure rate transformation procedure to be used.

TASK SECTION 101

SOFTWARE RELIABILITY PREDICTION BASED ON APPLICATION

101.1 PURPOSE

The purpose of task 101 is to provide a method for predicting a baseline software reliability.

101.2 DOCUMENTS REFERENCED

See Task Section 100.

101.3 GENERAL PROCEDURES

APPLICATION TYPE (A)

Using Data Collection Procedure 1 (Appendix B) and the corresponding Worksheet 0 (Appendix C), identify which application the subject software represents. Assign the corresponding fault density to A using Table TS101-1. Use the average fault density column.

An initial RP = A.

101.4 DETAIL TO BE SPECIFIED BY THE PROCURING AUTHORITY.

a. Statement of reliability requirements.

b. Define the software component level for prediction (different levels may be specified for each life cycle phase).

c. Define life cycle phases to be covered and prediction milestones.

d. Identify data collection procedures (see Appendix B of this Guidebook).

e. Identify fault density/failure rate transformation procedure to be used.

TABLE TS101-1
BASELINE FAULT DENSITY
AND FAILURE RATE VALUES

	FAULT DENSITY				FAILURE RATE		
APPLICATION	No. of SYSTEMS	TOTAL LINES OF CODE	AVG FAULT DENSITY BY SYSTEM	STD DEV FAULT DENSITY	NO. OF SYSTEMS	OPERATIONAL FAILURE RATE	STD DEV FAILURE RATE
AIRBORNE	7	540,617	.0128	.0094	1	.08*	--
STRATEGIC	21	1,793,831	.0092	.014	5	.0109	.012
TACTICAL	5	88,252	.0078	.0061	5	.108	.0054
PROCESS CONTROL	2	140,090	.0018	.0003	--	--	--
PRODUCTION CENTER	12	2,575,427	.0085	.0095	4	.198	348
DEVELOPMENTAL	4	97,435	.0123	.0093	1	21*	--
TOTAL/AVERAGE	59	5,235,652	.0094	.011	14	.1	.19

*These two values are failure rates during test, not include in total

TASK SECTION 102
SOFTWARE RELIABILITY PREDICTION BASED ON DEVELOPMENT ENVIRONMENT

102.1 PURPOSE

The purpose of task 102 is to modify the baseline software reliability prediction calculated in task 101 based on the type of development environment.

102.2 DOCUMENTS REFERENCED

See task Section 100.

102.3 GENERAL PROCEDURES

DEVELOPMENT ENVIRONMENT (D)

Using Data Collection Procedure 2 (Appendix B) and the corresponding Worksheet 1 (Appendix C), identify which class of development environment is being used.

Three classes of development environments have been provided:

- Organic -- Software is being developed by a group that is responsible for the overall application (e.g., flight control software being developed by a manufacturer of flight control systems);
- Semi-detached -- The software developer has specialized knowledge of the application area, but is not part of the sponsoring organization (e.g., network control software being developed by a communications organization that does not operate the target network); and
- Embedded -- Software that frequently has very tight performance constraints being developed by a specialist software organization that is not directly connected with the application (e.g., surveillance radar software being developed by a group within the radar manufacturer, but not organizationally tied to the user of the surveillance information).

The baseline is the semi-detached environment. It is expected that the organic environment will generate software of lower fault density and the embedded environment software of greater fault density. Based on the selection, assign the corresponding value, D_O, to D from Table TS102-1.

$$D = D_O$$

TABLE TS102-1. DEVELOPMENT ENVIRONMENT METRICS

ENVIRONMENT	METRIC (FAULT DENSITY MULTIPLIER) D_O
ORGANIC	.76
SEMI-DETACHED	1.0
EMBEDDED	1.3

If more specific data about the environment is available, calculate D using the checklist in Data Collection Procedure 2 (Appendix B) for Development Environment. This modified approach is based on specific organizational/personnel considerations, methods used, documentation to be produced, and tools to be used. Recalculate D as:

$$D = D_m$$

where D_m is calculated from the following:

$$D_m = (.109\ D_C + -.04)/.014, \text{ Embedded}$$

$$D_m = (.008\ D_C + .009)/.013, \text{ Semi-detached}$$

$$D_m = (-.018\ D_C + -.003)/.008. \text{ Organic}$$

where D_C = the ratio of methods and tools checked divided by the total number of methods and tools in the checklist in worksheet 1. D_C is a ratio of methods + tools checked/38. D_C is a number between 0 and 1. D_m should never be less than .5 or greater than 2. If the calculations result in a number less than .5, set D_m = .5. If the calculations result in a number greater than 2, set D_m = 2.

$$D = D_m$$

PREDICTION

An updated prediction is calculated by:

$$RP = A*D$$

102.4 DETAIL TO BE SPECIFIED BY THE PROCURING AUTHORITY.

a. Identify data collection procedures (see Appendix B of this Guidebook).

b. Specify whether a generic or detail development environment factor is to be generated.

TASK SECTION 103

SOFTWARE RELIABILITY PREDICTION BASED ON SYSTEM/SUBSYSTEM LEVEL SOFTWARE CHARACTERISTICS

103.1 PURPOSE

The purpose of task 103 is to modify the baseline software reliability prediction calculated in task 101 based on the software characteristics as they evolve during the requirements and design phases of a development.

103.2 DOCUMENTS REFERENCED

See Task Section 100.

103.3 GENERAL PROCEDURES

SOFTWARE CHARACTERISTICS (S)

The Software Characteristic metric, S, is a product (composite) of two submetrics;

$$S = S1*S2$$

Each one of which is in turn the product of several simple metrics as shown below:

REQUIREMENTS AND DESIGN REPRESENTATION METRIC S1 = SA*ST*SQ

Anomaly Management (SA)	- optional
Traceability (ST)	- optional
Quality Review Results (SQ)	- optional

SOFTWARE IMPLEMENTATION METRIC S2 = SL*SM*SX*SR

Language (SL)	- recommended
Modularity (SM)	- optional
Complexity (SX)	- recommended
Standards Review (SR)	- recommended

S1 is described in this task. S2 is described in task 104.

Note: These metrics and their corresponding impact on the reliability prediction are based on data collected from

several projects. Those identified above as recommeded have exhibited consistency good predictive results. Those listed as optional provide the reliability engineer additional information upon which to base the prediction but because either their predictive qualities have been inconsistent or they are based on a limited sample size, they are not recommended.

A description of the procedures for calculating these metrics follows.

Anomaly Management (SA) - optional

- Apply Data Collection Procedure 3 (Appendix B) and the corresponding Worksheet 2 in Appendix C to the Requirements and Design Specifications of the subject project. Answer all questions related to Anomaly Management (AM.1 through AM.7).

- Calculate the Anomaly Management Metric using the following equation:

$$\begin{aligned} SA &= .9 \text{ IF } AM > .6 \\ &= 1 \text{ IF } .6 \geq AM \geq .4 \\ &= 1.1 \text{ IF } AM < .4 \end{aligned}$$

  where AM equals the score received using the worksheet.

- Note that SA is applicable at SPR, PDR, CDR, and during coding. The appropriate worksheet should be used depending on the reliability prediction milestone to calculate the AM metric.

Traceability (ST) - optional

- Apply Data Collection Procedure 4 (Appendix B) and the corresponding Worksheet 3 in Appendix C to the Requirements and Design Specifications and code of the subject project. Answer the traceability questions. If unable to answer, use following substeps:

  - Itemize all specific requirements in Requirements Specification.

  - Count the number of individual requirements (NR). See Data Collection Procedure 7 in Appendix B.

  - Review Design Specification and identify specific design statements that represent the fulfillment of a specific itemized requirement (a requirements derivitive).

  - Count the number of requirements not addressed by design that should have been (DR).

- Calculate ST as follows:

$$ST = 1.1 \text{ IF } \frac{NR - DR}{NR} < .9$$

$$= 1 \text{ IF } \frac{NR - DR}{NR} > .9$$

Quality Review Results (SQ) - optional

- Apply Data Collection Procedure 5 (Appendix B) and the corresponding Worksheet 10, in Appendix D of this report to the Requirements and Design Specifications of the subject project. Answer all questions related to Accuracy (AC.1), Completeness (CP.1), Consistency (CS.1, CS.2) and Autonomy (AU.1, AU.2).

- Identify discrepancies. Total the number of discrepancies identified (DR). See Data Collection Procedure 12 in Appendix B and Worksheet 5 in Appendix C.

- Count the number of individual requirements (NR). See Data Collection Procedure 6 (Appendix B) and Worksheet 3 in Appendix C.

- Calculate the SR metric using the following equation:

During Requirements and Preliminary Design:

$$SQ = 1.1 \text{ IF } \frac{DR}{NR} > .5$$

$$1 \text{ IF } \frac{DR}{NR} < .5$$

During Detailed Design:

$$SQ = 1.1 \text{ if } \frac{DR}{NM} > .5$$

$$= 1 \text{ if } \frac{DR}{NM} < .5$$

PREDICTION

If these optional metrics are applied, then an updated prediction is calculated by:

RP - A*D*S1

103.4 DETAIL TO BE SPECIFIED BY THE PROCURING AGENCY

a. Identify data collection procedures (see Appendix B).

b. Specify that requirements shall be traced throughout dvelopment.

TASK SECTION 104

SOFTWARE RELIABILITY PREDICTION BASED ON CSC/UNIT LEVEL CHARACTERISTICS

104.1 PURPOSE

The purpose of task 104 is to modify the baseline software reliability prediction calculated in task 101 based on the software characteristics as they evolve during the coding phase of a development.

(This task can only be specified if Task 103 is also specified. See Task 103 for algorithm for combining the individual factors computed in Task 104.)

104.2 DOCUMENTS REFERENCED

See Task Section 100

104.3 GENERAL PROCEDURES

For each of the following metrics calculate their influence on software reliability. Note that some metrics are recommended and some are optional. See Task 103 (Note) for an explanation of the optional metrics.

Language Type (SL) - recommended

- Identify the total number of executable lines of code (SLOC) in the system (estimated or actual), the number of assembly language lines (ALOC), and the number of higher order language lines (HLOC). Use data collection procedure 6 and 8 in Appendix B and Data Collection Worksheet 4 in Appendix C.

- Use the following equations to calculate the language metric

$$SL = HLOC/SLOC + 1.4 * ALOC/SLOC$$

Modularity (SM) - optional

- Count the number of modules in the system (NM) and the lines of executable code in each module (MLOC(i)). Use data collection procedure 9 in Appendix B and Data Collection Worksheet 4 in Appendix C.

- Table TS104-2 illustrates the impact on the predicted reliability by the number of modules in each size category.

- Use the following equation to calculate modularity:

SM = (u*.9 + w + x*2)/NM

where NM = u + w + x

TABLE TS104-2. MODULE CATEGORIES

SIZE CATEGORY	NUMBER OF MODULES IN SIZE CATEGORY	MODULARITY METRIC (SM)
LOC < 200	u	.9
200 < LOC < 3000	w	1
3,000 < LOC	x	2

Complexity (SX) - recommended

- Apply McCabe's complexity measure to each module in the system (sx(i)). Use data collection procedure 10 in Appendix B and Data Collection Worksheet 4 in Appendix C.
- Use the following equation to derive system complexity multiplier:

SX = (1.5(a) + 1(b) + .8(c))/ NM

where
a = number of modules with a complexity greater than 20.

b = number of modules with a complexity between 7 and 20.

c = number of Modules with a complexity less than 7.

NM = the total number of modules = a + b + c.

Standards Review (SR) - recommended

- Apply data collection procedure 11 (Appendix B) and corresponding Worksheet (11) in Appendix D to the code. Answer all questions related to SI.1, SI.2, SI.4, SI.5, MO.1, MO.2.
- Identify the number of modules with problems (PR)

- Calculate this metric as follows:

$$SR = 1.5 \text{ if } \frac{PR}{NM} > .5$$

$$1 \text{ if } .50 > \frac{PR}{NM} > .25$$

$$.75 \text{ if } \frac{PR}{NM} < .25$$

PREDICTION

Based on the application of these metrics, an updated prediction is calculated by:

$$RP = A * D * S1 * S2$$

If optional metrics are not used then assign a value of 1 to their multiplier.

104.4 DETAIL TO BE SPECIFIED BY THE PROCURING AGENCY

a. Identify Data Collection Procedures (see Appendix B).

TASK SECTION 200
SOFTWARE RELIABILITY ESTIMATION

200.1 PURPOSE

The purpose of task 200 is to describe the general procedures for estimating software reliability during testing.

200.2 DOCUMENTS REFERENCED IN TASK SECTION 200

MIL-STD 2167A
RADC TR 83-11
RADC TR 84-53
MIL-STD 756B
MIL-STD 785B

200.3 GENERAL PROCEDURES

200.3.1 RELIABILITY MODEL

The general block diagrams applicable to software reliability prediction can also be used for software reliability estimation. For each block in the diagram (at a level where a block is a processing component like a computer), software reliability estimation is going to be based on performance results during test conditions.

Once software is executing its failure rate can be directly observed and a transformation is no longer required.

The failure rate of a program during test is expected to be affected by the amount of testing performed, the methodology employed, and the thoroughness of the testing. The following models are applicable to an estimation of the failure rate based on results from the test environment.

Estimating software reliability for a software component (whether it is at a system level where all software operates on one CPU or at a CSCI level with CSCI's operating on various CPU's) can be approached in two ways. The two approaches are described in the following paragraphs. Each requires observing the failure rate and testing time. Data collection procedures 12, 13, and 14 in Appendix B are used for measuring the software failure rate.

200.3.1.1 Reliability Estimation Modeling Approach 1

Several models have been suggested for relating failure experience to execution time (see RADC TR 83-11). The Musa model as an example, assumes that the failure rate is proportional to the number of faults in a segment, and that the number of faults is being reduced every time a failure is encountered (not necessarily one fault removed for every failure encountered). This leads to an exponential distribution of faults with

execution time, a one parameter distribution in which the scale parameter can be estimated by established methods. In spite of substantial evidence that the execution time to encounter a fault can vary by several orders of magnitude, the Musa model seems to yield acceptable results for the test and early operational phases. The general prediction and estimation methodology can be used with any other execution time based model (RADC TR 83-11).

The failure rate during test, F, is given by

$$F = L_0 \exp(- L1 * t)$$

where the amount of test time, t, is measured in terms of CPU-time, based on a 32-bit, 10 MIPS execution. L_0 and L1 are the scale parameters proportional to the fault density. Any of the models described in RADC TR 83-11 can be used to model the failure rate observed during testing. Once modeled, the time until an acceptable failure rate is achieved can be calculated and operational performance can be estimated.

200.3.1.2 Reliability Estimation Modeling Approach 2

This approach does not use the models described in 200.3.1.1. It uses the failure rate observed during testing and modifies that rate by parameters estimating the thoroughness of testing and the extent to which the test environment simulates the operational environment. This is the approach used in Task Section 201 of this guidebook.

The estimated failure rate then is

$$F = F_{T1} * T_1 \text{ or}$$

$$F_{T2} * T_2$$

where F_{T1} is the average observed failure rate during testing.
and F_{T2} is the observed failure rate at end of test.

$$T_1 = .02 * T$$

$$T_2 = .14 * T$$

$$\text{and } T = TE * TM * TC$$

where TE is a measure of test effort, TM is a measure of test methodology and TC is a measure of test coverage. Definitions and calculation of these measures is in task 201.

The most significant aspect of the test environment is that it represents a deliberate increase in the potential for detecting

failure by:

- Construction of test cases that represent a much higher variability of the input and control states than is expected in operation;
- Close scrutiny of the computer output so that practically all failures that do occur are detected; and
- Creating a high workload, particularly for stress tests, which increases the probability of failure.

Empirical data has shown that the average rate during test is 50 times greater than during operation and the failure rate at end of test is approximately 7 times that observed during operation. The .02 term in the T1 equation and the .14 term in the T2 equation above represent these observations.

The stress the operating environment will have on the software also must be taken into account. The basic failure rate relation for the initial operating environment is similar to that developed for the test environment except that the operating environment metric, E, replaces the test environment factor.

$$F = F_{T2} * T_2 * E$$

Here T_2 is .14 and the baseline value for E is 1. Modifiers for the operating environment factor arise from variability of the data and control states (EV) and from workload (EW) as discussed in task 202.

200.4 DETAILS TO BE SPECIFIED BY PROCURING ACTIVITY

a. Requirement of tasks 201 and 202.

b. Definition of test phases to be used.

c. Definition of qualification test requirements.

d. Statement of requirement for discrepancy reporting.

TASK SECTION 201

RELIABILITY ESTIMATION FOR TEST ENVIRONMENT

201.1 PURPOSE

The purpose of task 201 is to describe the procedures for estimating what the operational reliability will be based on observed failure rate during testing.

201.2 DOCUMENTS REFERENCED IN TASK 201

See task Section 200.

201.3 GENERAL PROCEDURES

The influence the test environment has on the estimate of failure rate is described by three parameters described in the following paragraphs.

Several characteristics of the test environment are accounted for in the estimation of reliability. The observed failure rate may not accurately represent what the operational reliability will be because:

- The test environment does not accurately represent the operational environment,
- The test data does not thoroughly exercise the system thereby leaving untested many segments of the code,
- The testing techniques employed do not thoroughly test the system, and
- The amount of testing time does not allow for a thorough test of the system.

These characteristics are taken into account by the metrics to be discussed in this paragraph. In each case the metrics will be in the form of a multiplier, the product of all of these to be used to adjust the observed failure rate (F_T) up or down depending on the level of confidence in the representativeness and thoroughness of the test environment.

DETERMINATION OF FAILURE RATE DURING TEST

Using Data Collection Procedures 12, 13 and 14 in Appendix B and Data Collection Worksheets 5 and 6 in Appendix C, calculate the current average failure rate during testing (F_{T1}). The average failure rate during testing can be calculated at anytime during formal testing. It is based on the current total number of discrepancy reports recorded and the current total amount of test

operation time expended. It is expected that the failure rate will vary widely depending on when it is computed. For more consistent results, average failure rates should be calculated for each software test phase: CSC Integration and Testing, CSCI Testing; and, if required, for each system test phase: Systems Integration and Testing, and Operational Testing and Evaluation.

If the estimation is being made at the end of testing prior to deployment of the system, the estimation can be based on the failure rate observed at the end of CSCI testing (F_{T2}). The failure rate calculation in this case is based on the number of discrepancy reports recorded and amount of computer operation time expended during the last three test periods of CSCI testing. Data Collection Procedure 14 should be used to calculate F_{T1} and F_{T2}.

ESTIMATE SOFTWARE RELIABILITY

Using the currently observed average failure rate during testing, an estimate of the operational failure rate can be calculated by:

$$F = F_{T1} * T_1$$

where $T_1 = .02 * TE * TM * TC$

The multipliers TE, TM and TC are determined as follows:

Test Effort (TE) - optional

- Three alternatives are provided for measuring test effort. The choice will primarily depend on availability of data. Data Collection Procedure 15 is in Appendix B and Worksheet 6 in Appendix C aid in the collection and calculation of this metric.
  - The preferred alternative is based on the labor hours expended on software test. As a baseline, 40% of the total software development effort should be allocated to software test. A higher percentage indicates correspondingly more intensive testing, a lower percentage less intensive testing.
  - The second alternative utilizes funding instead of labor hours.
  - The third alternative is the total calendar time devoted to test.
- The metric, TE, will be set based on observing these three characteristic during the validation phase of the project. Use data collection procedure 15 (Appendix B). The three characteristics impact TE as follows:

if $40/AT \leq 1$

where AT - the percent of the development effort devoted to testing, then TE - .9

or if $40/AT \geq 1$

where AT - the percent of the development schedule devoted to testing, then set TE - 1.0.

Test Methodology (TM) - Optional

- The test methodology factor, TM, represents by the use of test tools, and test techniques. In most cases the tools, and techniques are being operated by a staff of specialists who are also aware of other advances in software test technology.

- In the Software Test Handbook, RADC TR 84-57, a technique to determine what tools and techniques should be applied to a specific application is provided. That technique is illustrated in Figure TS201-1 and results in a recommended set of testing techniques and tools. The approach is to use that recommendation to evaluate the techniques and tools applied on a particular development. Use Data Collection Procedure 16 and Worksheet 7. This evaluation will result in a score that will be the basis for this metric as follows:

  TM - .9 for $TU/TT > .75$

  TM - 1 for $.75 > TU/TT > .5$

  TM - 1.1 for $TU/TT < .5$

  where TU is the number of tools and techniques used and TT is the number recommended.

Test Coverage (TC) - Recommended

- This metric assesses how thoroughly the software has been exercised during testing. If all of the code has been exercised then there is some level of confidence established that the code will operate reliably during operation. Typically however, test programs do not maintain this type of information and a significant portion (up to 40%) of the software (especially error handling code) may never be tested. Use data collection procedure 17 and Data Collection Worksheet 8.

- This metric can be calculated in three ways depending on

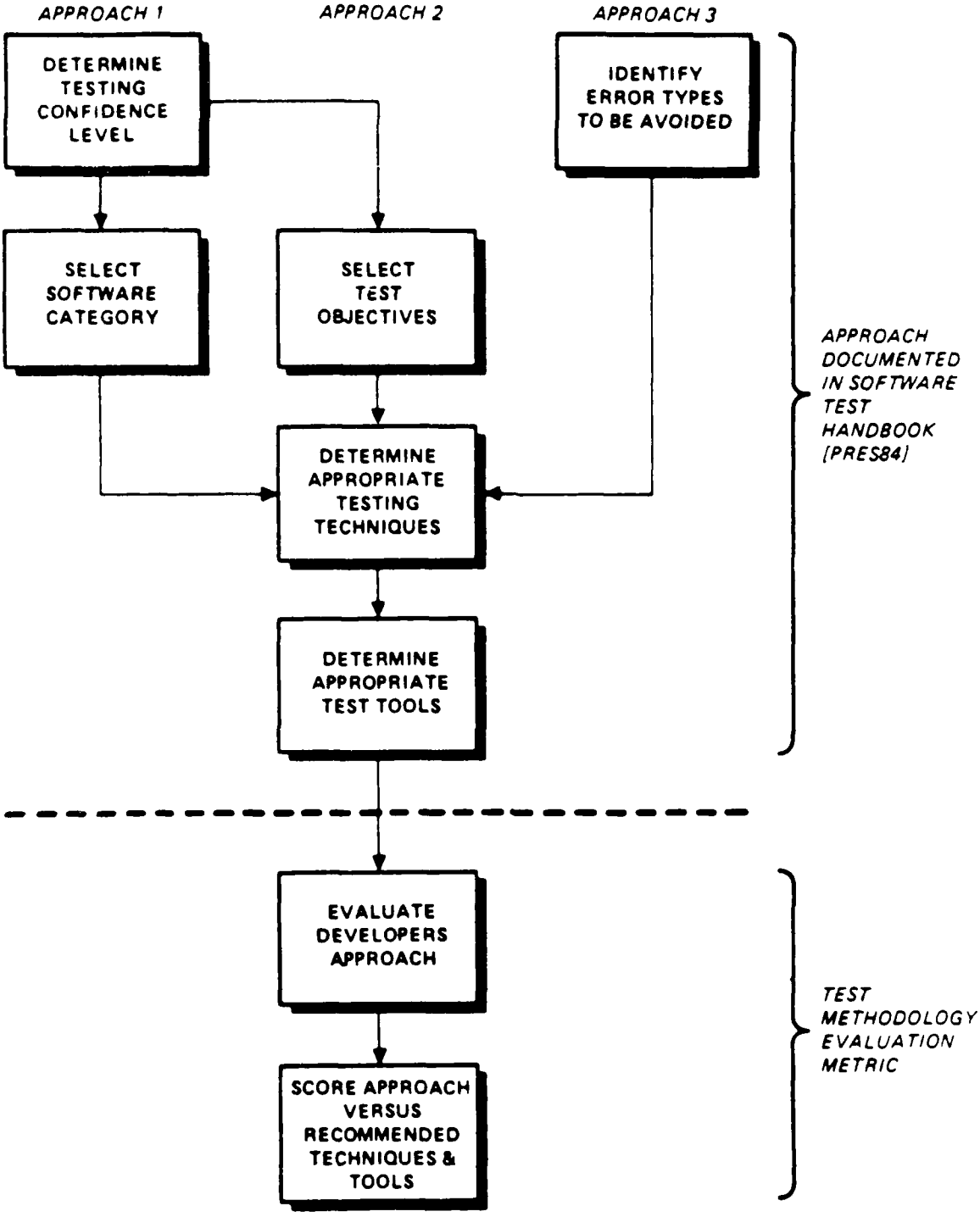


FIGURE TS201-1 TEST METHODOLOGY ASSESSMENT APPROACH

the phase of testing as follows:

$$TC = 1/VS$$

where VS = VS1 during CSC testing
= VS2 during CSC integration and test
= VS3 during CSCI testing

and

VS1 = (PT/TP + IT/TI)/2
where PT = execution paths tested
TP = total execution paths
IT = input tested
TI = total number of inputs

VS2 = (MT/NM + CT/TC)/2
MT = units tested
NM = total number of units
CT = interfaces tested
TC = total number of interfaces

VS3 = RT/NR
RT = Requirements tested
NR = total number of requirements

An updated reliability estimation can be made using these multipliers at the end of test by using:

$$F = F_{T2} * T_2$$

where $T_2 = .14 * TE*TM*TC$

A comparision of the predicted fault density (determined using Tasks 101 through 104) with the actual fault density realized can be made. Using Data Collection procedure 7 in Appendix B, the fault density realized is the number of discrepancy reports reported during testing divided by the total number of executable lines of code in the system. A comparison of the predicted failure rate, transformed from the predicted fault density, can also be made with the estimated failure rate calculated in this task. Significant variation in these values suggests that analyses be conducted to evaluate the differences. Consistent values suggests accurate predictions and estimations.

201.4 DETAILS TO BE SPECIFIED BY THE PROCURING AUTHORITY.

a. Define the software component level for estimation (different levels may be specified for each life cycle phase)

b. Define life cycle phases to be covered and estimation milestones.

c. Identify data collection procedures (see Appendix B of this Guidebook)

TASK SECTION 202

SOFTWARE RELIABILITY ESTIMATION FOR OPERATING ENVIRONMENT

202.1 PURPOSE.

The purpose of task 202 is to describe the procedures for estimating what the operational reliability will be based on estimates of the operational environment and the observed failure rate at the end of test.

202.2 DOCUMENTS REFERENCED IN TASK SECTION 202

See Task Section 200.

202.3 GENERAL PROCEDURES

Two factors are accounted for in estimating the failure rate for the operational environment: the workload expected and the input variability. These both represent expected stress on the system.

ESTIMATE SOFTWARE RELIABILITY

Using the end of test failure rate (see Task 201 and Data Collection Procedure 14 in Appendix B), F_{T2}, an estimate of the operational reliability is calculated as follows:

$$F = F_{T2} * T_2 * E$$

where F_{T2} is the failure rate at end of test

$T_2 = .14$

E = EV*EW, modifiers representing stress of input variability, EV, and workload, EW.

The modifiers are calculated as follows:

Variability of Data and Control States (EV) - Recommended

- Software that is delivered for Air Force use should be essentially fault free for nominal data and control states, i.e., where an input is called for, an input fully compliant with the specification will be present; when an output is called for, the channel for receiving the output will be available. A major factor in the occurrence of failures, and therefore affecting the failure rate, will be the variability of input and control states.

- The frequency of exception conditions as a measure of variability is used here. Exception states include:

  - Page faults, input/output operations, waiting for

completion of a related operation -- the frequency of all of these is workload dependent and the effect on software reliability is discussed in the next paragraph;

- Response to software deficiencies such as overflow, zero denominator, or array index out of range; and

- Response to hardware difficulties such as parity errors, error correction by means of code, or noisy channel.

The last two of these are combined in the input variability modifier for the operating environment, EV. Data illustrated in Table TS202-1, indicates that approximately 1,000 exception conditions of the latter two types were encountered in 5,000 hours of computer operation. A value of 0.2 exception conditions per computer hour has therefore been adopted as the baseline, to be equated to unity. Because failures may arise even if no exception conditions at all are encountered, it is desirable to bias the modifier to a small positive value. The suggested form is

$$EV = 0.1 + 4.5EC$$

where EC is the number of exception conditions per hour. For EC = 0.2, EV = 1. Use Data Collection Procedure 18 (Appendix B).

Workload (EW) - Recommended

Significant effects of workload on software failure rate have been reported. The hazard function, the incremental failure rate due to increasing workload, ranges over two orders of magnitude.

For military applications, workload effects can be particularly important. During time of conflict, the workloads can be expected to be exceptionally heavy, causing the expected failure rate to increase, and yet at that same time a failure can have the most serious consequences. Hence, predictions of failure rates that do not take workload effects into account fail to provide the information that Air Force decision makers need.

The mechanism by which workload increases the failure rate is not completely known, but it is generally believed to be associated with a high level of exception states, such as busy I/O channels, long waits for disk access, and possibly increased memory errors (due to the use of less frequently accessed memory blocks). Data show that the highest software (and also hardware) failure rates were experienced during the hours when the highest levels of exception handling prevailed.

Details of workload effects on software failure rate are still a research topic, and no specific work in that area has been included in this Guidebook. The estimations will be based on published work, such as Figure TS202-1. The quantity plotted

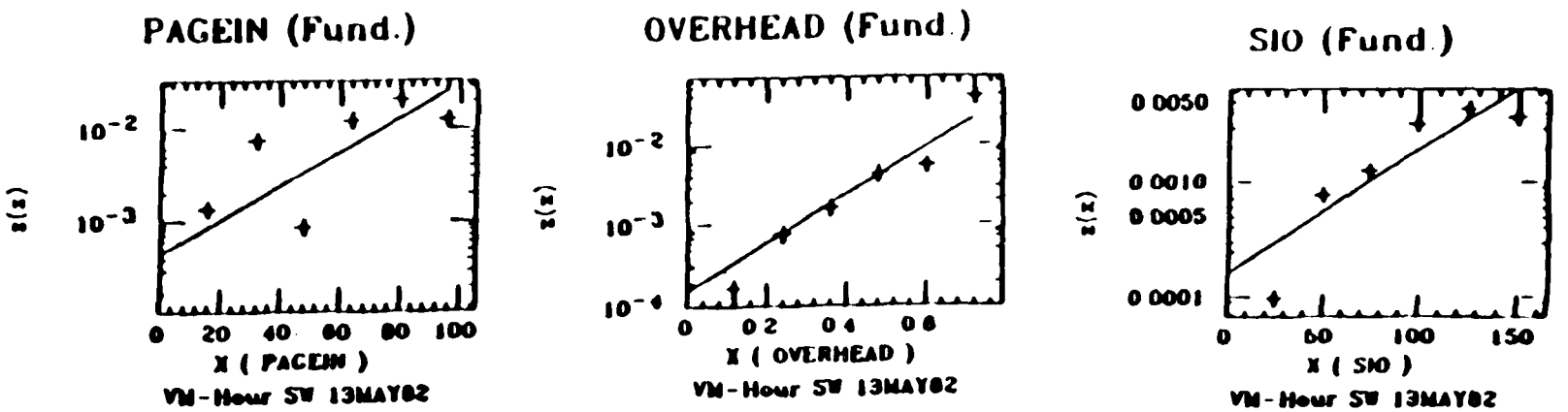


FIGURE TS202-1 EFFECT OF WORKLOAD ON SOFTWARE HAZARD

along the vertical axis is the inherent load hazard, z(x), defined as:

> Probability of failure in workload interval (x, x+ delta x)/Probability no failure in interval (0,x).

It measures the incremental risk of failure involved in increasing the workload from x to x+delta x.

The horizontal axis shows three different measures of workload:

- Virtual memory paging activity, number of pages read per second (PAGEIN);
- Operating system overhead, fraction of time not available for user processes (OVERHEAD); and
- Input/output activity, number of non-spooled input/output operations started per second (SIO).

These graphs provide an option of estimating workload effects by any of the indicators of workload used here. The fraction of overhead usage is probably the most commonly obtainable quantity. From a practical point of view, before a computer installation becomes operational, the fraction of capacity to be used at maximum expected workload is probably the only indication of this factor that will be available early in the development. Data Collection Procedure 19 in Appendix B and Worksheet 9 in Appendix D should be referenced.

The workload metric takes the form

$$EW = ET/(ET-OS)$$

where ET = Executive Time
OS = Operating System Overhead time

202.4 DETAIL TO BE SPECIFIED BY THE PROCURING AUTHORITY.

a. Define the software component level for estimation (different levels may be specified for each life cycle phase). CSCI level is recommended.

b. Define life cycle phases to be covered and estimation milestones.

c. Identify data collection procedures (see Appendix B of this Guidebook).

APPENDIX A

DEFINITIONS AND TERMINOLOGY

This appendix presents definitions of the principal terms and concepts used in this report. Where possible, the definitions are taken from established dictionaries or from the technical literature. Where a rationale for the selection or formulation of a definition seems desirable, it is provided in an indented paragraph following the definition. The sources for the definitions will be found in the list of references at the end of this Guidebook.

ERROR - A discrepancy between a computed observed, or measured value or condition and the true, specified, or theoretically correct value or condition. [ANSI81]

> This definition is listed as (1) in the American National Dictionary for Information Systems. Entry (2) in the same reference states that error is a "Deprecated term for mistake". This is in consonance with [IEEE83] which lists the adopted definition as (1) and lists as (2) "Human action that results in software containing a fault. Examples include omission or misinterpretation of user requirements in a software specification, incorrect translation or omission of a requirement in a design specification. This is not a preferred usage."

FAILURE - The inability of a system or system component to perform a required function with specified limits. A failure may be produced when a fault is encountered. [IEEE83]

> This definition is listed as (2) in the cited reference which lists as (1) "The termination of the ability of a functional unit to perform its required function" and as (3) "A departure of program operation from program requirements". Definition (1) is not really applicable to software failures because these may render an incorrect value on one iteration but correct values on subsequent ones. Thus, there is no termination of the function in case of a failure. Definition (3) was considered undesirable because it is specific to the operation of a computer program and a more system-oriented terminology is desired for the purposes of this study.

FAULT - An accidental condition that causes a functional unit to fail to perform its required function. [IEEE83]

> This definition is listed as (1) in the cited reference which lists as (2) "The manifestation of an error (2) in software. A fault, if encountered, may cause a failure". Error (2) is

identified as synonymous with "mistake". Thus this definition states that a fault is the manifestation in software of a (human) mistake. This seems less relevant than the identification of a fault as the cause of a failure in the primary definition. It is recognized that the presence of a fault will not always or consistently cause a unit to fail since the presence of a specific environment and data set may also be required (see definition of software reliability).

MISTAKE - A human action that produces an unintended result. [ANSI81]

SOFTWARE QUALITY FACTOR - A broad attribute of software that indicates its value to the user, in the present context equated to reliability. Examples of software quality factors are maintainability, portability, as well as reliability. May also be referred to simply as factor or quality factor. [Based on MCCA80]

SOFTWARE QUALITY METRIC - A numerical or logical quantity that measures the presence of a given quality factor in a design or code. An example is the measurement of size in terms of lines of executable code (a quality metric). May also be referred to simply as metric or quality metric. A single quality factor may have more than one metric associated with it. A metric typically is associated with only a single factor. [Based on MCCA80]

SOFTWARE RELIABILITY - The probability that software will not cause the failure of a system for a specified time under specified conditions. The probability is a function of the inputs to and use of the system as well as a function of the existence of faults in the software. The inputs to the system determine whether existing faults, if any, are encountered. [IEEE83]

This definition is listed as (1) in the IEEE Standard Glossary. An alternate definition, listed as (2), is "The ability of a program to perform a required function under stated conditions for a specified period of time." This definition is not believed to be useful for the current investigation because (a) it is not expressed as a probability and therefore cannot be combined with hardware reliability measures to form a system reliability measure, and (b) it is difficult to evaluate in an objective manner. The selected definition fits well with the methodology for software reliability studies which will be followed in this study, particularly in that it emphasizes that the presence of faults in the software as well as the inputs and conditions of use will affect reliability.

SOFTWARE RELIABILITY MEASUREMENT - The life-cycle process of establishing quantitative reliability goals, predicting, measuring, and assessing the progress and achievement of those goals during the development, testing, and O&M phases of a software system.

SOFTWARE RELIABILITY PREDICTION - A numerical statement about the reliability of a computer program based on characteristics of the design or code, such as number of statements, source language or complexity. [HECH77]

Software reliability prediction is possible very early in the development cycle before executable code exists. The numeric chosen for software reliability prediction should be compatible with that intended to be used in estimation and measurement.

SOFTWARE RELIABILITY ESTIMATION - The interpretation of the reliability measurement on an existing program (in its present environment, e.g., test) to represent its reliability in a different environment (e.g., a later test phase or the operations phase). Estimation requires a quantifiable relationship between the measurement environment and the target environment. [HECH77]

The numeric chosen for estimation must be consistent with that used in measurement.

SOFTWARE RELIABILITY ASSESSMENT - Generation of a single numeric for software reliability derived from observations on program execution over a specified period of time. Defined sections of the execution will be scored as success or failure. Typically, the software will not be modified during the period of measurement, and the reliability numeric is applicable to the measurement period and the existing software configuration only. [HECH77]

The statement about not modifying the software during the period of measurement is necessary in order to avoid committing to a specific model of the debugging/reliability relation. In practice, if the measurement interval is chosen so that in each interval only a small fraction of the existing faults are removed, then the occurrence of modifications will not materially affect the measurement.

PREDICTIVE SOFTWARE RELIABILITY FIGURE-OF-MERIT (RP) - A reliability number (fault density) based on characteristics of the application, development environment, and software implementation. The RP is established as a baseline as early as the concept of the system is determined. It is then refined based on how the design and implementation of the system evolves.

RELIABILITY ESTIMATION NUMBER (RE) - A reliability number (failure rate) based on observed performance during test conditions.

FUNCTION - A specific purpose of an entity or its characteristic action. [ANSI81] A subprogram that is invoked during the evaluation of an expression in which its name appears and that returns a value to the point of invocation. Contrast with

subroutine. [IEEE83]

MODULE - A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to, or output from, an assembler, compiler, linkage editor, or executive routine. [ANSI81] A logically separable part of a program. [IEEE83]

SUBSYSTEM - A group of assemblies or components or both combined to perform a single function. [ANSI73] In our context, a subsystem is a group of modules interrelated by a common function or set of functions. Typically identified as a Computer Program Configuration Item (CPCI) or Computer Software Configuration Item (CSCI). A collection of people, machines, and methods organized to accomplish a set of specific functions. [IEEE83] An integrated whole that is composed of diverse, interacting, specialized structures and subfunctions. [IEEE83] A group or subsystem united by some interaction and interdependence, performing many duties but functioning as a single unit. [ANSI73]

SYSTEM - In our context, a software system is the entire collection of software modules which make up an application or distinct capability. Along with the computer hardware, other equipment (such as weapon or radar components), people and methods the software system comprises an overall system.

APPENDIX B

DATA COLLECTION PROCEDURES

This appendix contains Data Collection Procedures. These procedures describe what data must be collected to use the Software Reliability Prediction and Estimation Procedures described in Section 5 of this report. Complementing these procedures are the actual worksheets contained in Appendix C and D. The intended process then is for reliability engineers to use the worksheets in conjunction with these data collection procedures to collect data. That data will then be used when the engineer or analyst uses the prediction and estimation algorithms to determine a reliability number. An index of data collection procedures is provided in Table B-1. A cross reference to the Worksheets in Appendix C and D is in Table B-2.

The utility of the metrics is based on their representation of the characteristics identified and the correlation or affect of these characteristics on software reliability. There is, however, another important aspect to the utility of the metrics. That is the economy of their use, i.e., the cost of collecting the data to calculate the metrics is an important consideration. Automated collection tools are essential for many of the measurements. Some measures, such as the ones which simply require classification, are easy to collect.

This appendix contains data collection procedures for all data required to calculate each metric. The procedures follow the following format:

PROCEDURE OUTLINE

1. Title: Identifies metric or data element this procedure relates to.

2. Prediction or Estimation Parameters Supported: Identifies the higher level metric this procedure relates to.

3. Objectives: Objective of the title metric.

4. Overview: Provides overview of this metric.

5. Assumptions/Constraints: Describes any assumptions or constraints related to this metric.

6. Limitations: Describes any limitations to using the

procedure or metric.

7. Applicability: Describes when the metric can be applied during the software life cycle.

8. Required Inputs: Identifies the required data for calculating the metric.

9. Required Tools: Identifies any required tools needed for data collection.

10. Data Collection Procedures: Provides step by step guidance on collecting the appropriate data.

11. Outputs: Describes output of procedure.

12. Interpretation of Results: Provides guidance on interpreting the results.

13. Reporting: Provides any required reporting format.

14. Form: Identifies any applicable forms for data collection or metric calculation.

15. Potential/Plans for Automation: Describes potential and any known plans for automation of this data collection procedure.

16. Remarks: Allows any remarks/comments about metric.

The data required for these metrics is available during most DOD software development. Data collection is required. It involves applying worksheets to the typical documentation produced with MIL-STD 2167A, MIL-STD 490/483, and MIL-STD 1679 and automated tools to the code produced. It also involves collecting data during test.

TABLE B-1. DATA COLLECTION PROCEDURE INDEX

DATA COLLECTION PROCEDURE NUMBER	RELATED METRIC	RELATED PROCEDURES
1	Application Type	-
2	Development Environment	-
3	Anomaly Management	-
4	Traceability	-
5	Quality Review	6,12
6	Size Estimation	7
7	Fault Density	1,2,3,4,5,6
8	Language	-
9	Modularity	8
10	Complexity	6,8,9
11	Standards Review	12
12	Discrepancy Reports	5,11
13	Execution Time	16
14	Failure Rate	14,15,7
15	Test Effort	-
16	Test Methodology	-
17	Test Coverage	-
18	Input Variability	-
19	Workload	-

PROCEDURE NO. 1

1. Title: Application Type (A)

2. Prediction or Estimation Parameter Supported: Application Type (A)

3. Objectives: At the system level categorize the system application according to the application and time dependence schemes identified in Worksheet 0. At a CSCI level, if possible, categorize the software by function.

4. Overview: Manual inspection of documentation to determine the type of system according to preceding classifications. This determination can be made at the Concept Definition phase.

5. Assumptions/Constraints: Ambiguities or other difficulties in applying this scheme should be resolved in favor of the dominant or most likely classification.

6. Limitations: None

7. Applicability: Identify Application Type at project initiation. Metric worksheets require update of information at each major review. It should not change.

8. Required Inputs: Statement of Need (SON), Required Operational Capability (ROC), or system requirements statement should indicate application type.

9. Required Tools: Visual inspection of documentation.

10. Data Collection Procedures: Functional description of system extracted from documentation and matched with an application area.

11. Outputs: A baseline fault density, A, will be associated with each Application Type.

12. Interpretation of Results: Application type may be used early in the development cycle to predict a baseline fault density. These rates are then modified as additional information concerning the software becomes available.

13. Reporting: Application type, together with projected baseline fault density, is reported. The baseline rate should be made available to the prospective user to ensure that the user is aware of failure rates (or fault density) for this application and has provisions which will affect the characteristics of the specific software as they unfold during system development.

14. Forms: Use Metric Worksheet O.

15. Potential Plans for Automation: None

16. Remarks: None

PROCEDURE NO. 2

1. Title: Development Environment (D)

2. Prediction or Estimation Parameter Supported: Development Environment (D)

3. Objectives: Categorizes the development environment according to Boehm's [BOEH81] classification. Additional distinguishing characteristics derived from RADC TR 85-47 are also used.

4. Overview: In Boehm's classification the system is categorized according to environment as follows:

> Organic Mode -- The software team is part of the organization served by the program.
>
> Semidetached Mode -- The software team is experienced in the application but not affiliated with the user.
>
> Embedded Mode -- Personnel operate within tight constraints. The team has much computer expertise, but is not necessarily very familiar with the application served by the program. System operates within strongly coupled complex of hardware, software, regulations, and operational procedures.

A survey in RADC TR 85-47 revealed the following factors, were felt to have significant impact on the reliability software. They, therefore, provide a checklist for predicting the quality of software produced using them:

- Organizational Considerations
  - Separate Design and Coding
  - Independent Test Organization
  - Independent Quality Assurance
  - Independent Configuration Control
  - Independent Verification/Validation
  - Programming Team Structure
  - Educational Level of Team Members
  - Experience Level of Team Members
- Methods Used

- Definition/Enforcement of Standards
- Use of High Order Language (HOL)
- Formal Reviews (PDR, CDR, etc.)
- Frequent Walkthroughs
- Top-Down and Structured Approaches
- Unit Development Folders
- Software Development Library
- Formal Change and Error Reporting
- Progress and Status Reporting

• Documentation

- System Requirements Specification
- Software Requirements Specification
- Interface Design Specification
- Software Design Specification
- Test Plans, Procedures and Reports
- Software Development Plan
- Software Quality Assurance Plan
- Software Configuration Management Plan
- Requirements Traceability Matrix
- Version Description Document
- Software Discrepancy Reports

• Tools Used

- Requirements Specification Language
- Program Design Language
- Program Design Graphical Technique (Flowchart, HIPO, etc.)
- Simulation/Emulation

- Configuration Management
- Code Auditor
- Test Data Generator
- Test Driver
- Automated Verification System
- Data Flow Analyzer
- Automated Measurement Tools

The developmental environment should be described in the Software Development Plan. If it is not, it will be necessary to review product reports or to interview the software developers.

5. Assumptions/Constraints: Use of the Boehm metric assumes a single dimension along which software projects can be ordered, ranging from organic to embedded. Care must be taken to ensure that there is some allowance made for variations from this single-dimensional model -- e.g. when inexperienced personnel are working in an in-house environment. In such cases, the dominant or most important characteristic will be used.

The checklist developed from RADC TR 85-47 provides a rating for the developmental environment and process. Higher scores are assumed to be associated with more reliable software. However, this relationship is not likely to be linear (that is, it is not likely that each item on the checklist will increase reliability by an identical amount). Calibration of the score will be required during tests of the metrics. Current values are from a survey.

6. Limitations: The reliability of these metrics will be affected by the subjective judgments of the person collecting the data. Data concerning project personnel may not always be available after project completion, unless it has been specifically gathered for this purpose.

7. Applicability: The Development Environment will be indicated during the requirements phase and, combined with expected fault density/failure rates for the Application Area, can be used to obtain an early forecast of reliability.

8. Required Inputs: Information is extracted visually from requirements or specifications documentation.

9. Required Tools: Manual data extraction from existing documentation. A checklist is provided in the Data Collection Worksheet 1 in Appendix C.

10. Data Collection Procedures: Using the classification scheme and checklist in Metric Worksheet 1, use Software Development Plan to determine the Development Environment metric. Where appropriate information is not included in available documentation, it may be necessary to interview project personnel.

11. Outputs: Classification and completed checklist as indicated in paragraph 9 above (Metric inputs D_o and D_c).

12. Interpretation of Results: As a refinement, regression techniques can be used to obtain metric values for each of the indicated environments in the Boehm classification. These are combined with the score obtained from the Martin Marietta checklist to obtain the score for this factor.

13. Reporting: Where the predicted failure rate differs from specified or expected values, changes in the personnel mix, project organization, methodology employed, or other environmental factors may be required to improve predicted reliability or to reduce costs. Early reporting of this information will permit such changes to be made in a timely fashion.

14. Forms: Use Metric Worksheet 1

15. Potential/Plans for Automation: This factor will be obtained manually.

16. Remarks: None

PROCEDURE NO. 3

1. Title: Anomaly Management (SA)

2. Prediction or Estimation Parameter Supported: Software Characteristics

3. Objectives: The purpose of this procedure is to determine the degree to which a software system is capable of responding appropriately to error conditions and other anomalies.

4. Overview: This metric is based on the following characteristics:

-- Error Condition Control,

-- Input Data Checking,

-- Computational Failure identification and Recovery,

-- Hardware Fault Identification and Recovery,

-- Device Error Identification and Recovery, and

-- Communication Failure Identification and Recovery.

In general, it is assumed that the failure rate of a system will decrease as anomaly management, as measured by this metric, improves.

This metric requires a review of program requirements specifications, and designs to determine the extent to which the software will be capable of responding appropriately to non-normal conditions, such as faulty data, hardware failures, system overloads, and other anomalies. Mission-critical software should never cause mission failure. This metric determines whether error conditions are appropriately handled by the software, in such a way as to prevent unrecoverable system failures.

5. Assumptions/Constraints: Elements of this metric are obtained manually in checklist form. This metric assumes that system requirements and specifications contain sufficient information to support computation of the required values.

6. Limitations: By its very nature, an anomaly is an unforeseen event, which may not be detected by error-protection mechanisms in time to prevent system failure. The existence of extensive error-handling procedures will not guarantee against such failures, which may be detected during stress testing or initial trial implementation. However, the metric will assist in determining whether appropriate error procedures have been included in the system specifications and designs.

7. Applicability: Elements of this metric will be obtained throughout the software development cycle.

8. Required Inputs: This procedure requires a review of all system documentation and code.

9. Required Tools: No tools will be used in the collection of data for this metric. A checklist is provided in the Worksheets in Appendix D.

10. Data Collection Procedures: Data to support this metric will be collected during system development, Data must be obtained manually, through inspection of code and documentation.

11. Outputs: The measurement, AM, is the primary output of this procedure. In addition, reports of specific potential trouble areas, in the form of discrepancy reports, will be desirable for guidance of the project manager and the program supervisor.

12. Interpretation of Results: Anomaly conditions require special treatment by a software system. A high score for AM would indicate that the system will be able to survive error conditions without system failures.

13. Reporting: An overall report concerning anomaly management will be prepared. It should be noted that the cost of extensive error-handling procedures must he balanced against the potential damage to be caused by system failure. A proper balance of costs and benefits must be determined by project management; the purpose of this metric is to assist the manager in assessing these costs and benefits.

14. Forms: Metric Worksheet 2 in Appendix C.

15. Potential Plans for Automation: Information for this metric is obtained manually.

16. Remarks: Proper determination of this metric will require some imagination and intelligent judgment on the part of the reviewer. Since error conditions take a wide variety of forms, the reviewer should be experienced in developing error-resistant software.

PROCEDURE NO. 4

1. Title: Traceability (ST)

2. Prediction or Estimation Parameter Supported: Software Characteristics

3. Objectives: The purpose of this metric is to determine the relationship between modules and requirements. If this relationship has been made explicit, there is greater likelihood that the modules will correctly fulfill the requirements. It should be possible to trace module characteristics to the requirements.

4. Overview: This metric indicates whether a cross reference exists which relates functions or modules to the requirements.

5. Assumptions/Constraints: The intent of the metric requires an evaluation of the correctness or completeness of the requirements matrix. It is assumed that the existence of the matrix will have a positive effect upon reliability.

6. Limitations: To achieve the true intent of this metric, a sophisticated tool or requirements specification language must be used. In its simplest form, the metric can simply be a check to see if a cross-reference matrix exists.

7. Applicability: Traceability may be determined during the requirements and design phases of the software development cycle.

8. Required inputs: Requirements and design documentation should include a cross reference matrix.

9. Required Tools: No special tools are required, however, use of a formal requirements specification language, PDL, or traceability tool provides significant savings in effort to develop this metric.

10. Data Collection Procedures: Documentation is reviewed to determine the presence or absence of the cross reference matrix, to itemize requirements at one level and their fulfillment at another. Metric Worksheet 3 in Appendix C can be used.

11. Outputs: Problem Reports should be written for each instance that a requirement is not fulfilled at a lower level specification.

12. Interpretation of Results: The cross reference should be taken as an indication of software quality, in that the presence of the matrix will make it more likely that implemented software actually meets requirements. Identified traceability problems should be reviewed for significance.

13. Reporting: The project engineer should be made aware of the presence or absence of the stated cross reference, to determine whether contractual requirements have been met.

14. Forms: Discrepancy Reports should be generated for all instances of lack of traceability. Metric Worksheet 3 in Appendix C contain checklist items for this item.

15. Potential/Plans for Automation: Tools such as PSL/PSA, SREM, RTT, USE-IT assist in the determination of this metric.

16. Remarks: None

PROCEDURE NO. 5

1. Title: Quality Review (SQ)

2. Prediction or Estimation Parameter Supported: Software Characteristics

3. Objectives: This procedure consists of checklists to assess the following characteristics:

-- Standard design representation;

-- Calling sequence conventions;

-- Input/output conventions;

-- Data naming conventions;

-- Error handling conventions;

-- Unambiguous references;

-- All data references defined, computed, or obtained from all external source;

-- All defined functions used;

-- All conditions and processing defined for each decision point;

-- All defined and referenced calling parameters agree;

-- All problem reports resolved;

-- Accuracy analysis performed and budgeted to module;

-- A definitive statement of requirement for accuracy of inputs, outputs, processing, and constraints;

-- Sufficiency of math library;

-- Sufficiency of numerical methods;

-- Execution outputs within tolerances; and

-- Accuracy requirements budgeted to functions/-modules.

These are combined to form a metric, SQ, which represents how well these characteristics have been designed and implemented in

the software system.

4. Overview: This metric will be determined at the requirements analysis and design phases of a software development. The metric itself is the number of problems found during reviews of the requirements and design of the system. In order to make this metric relative among systems, this number is divided (normalized) by the number of functional requirements identified for the system.

5. Assumptions/Contraints: Formal problem reporting during requirements and design phases of software developments has been inconsistently performed in the past. Methodologies advocated in recent years and more disciplined contractual/Government requirements and standards now encourage this activity. Assumed in this metric is a significant effort to perform formal reviews. Techniques such as Design Inspections or walkthroughs are the mechanism through which problems will be identified. Use of the worksheets at Appendix D are also an alternative.

6. Limitations: The degree to which the requirements and design specifications are reviewed will influence the number of problems found. Consistent application of the worksheets at Appendix C as a QA technique will alleviate this limitation.

7. Applicability: The primary application of this metric is to the requirements phase and design phases of the software development.

8. Required Inputs: Requirements Specification, Preliminary Design Specification, Detailed Design Specification are required.

9. Required Tools: Checklists will be used in determining this metric.

10. Data Collection Procedures: Documentation will be reviewed at the end of each phase of the system development to determine the presence or absence of these characteristics.

Since this procedure assesses the quality at early stages of the development, it will require a comprehensive review of documentation. Detailed records must be maintained (Discrepancy Reports). Reviews will be performed using the worksheet in Appendix D.

11. Outputs: Reports of the current number of discrepancy reports (DR), together with detailed information for the project manager, will be prepared.

12. Interpretation of Results: To some extent, software will be incomplete throughout most of the development cycle, until the point at which all variables, operations, and control structures are completely defined. This metric serves, then, as a measure of progress. An incomplete software system by definition, is

unfinished.

13. Reporting: Detailed reports of problems should be furnished to the project manager and the software supervisor, to assist in determining the current status of software development.

14. Forms: Worksheet 10 at Appendix D will be required.

15. Potential/Plan for Automation: RADC-developed Automated Measurement System (AMS) provides checklists for use in reviewing documents.

16. Remarks: Determination of quality will require extensive review of documentation, and will thus be expensive. The extra cost may be justified if the information obtained can be used to correct faults as they are uncovered.

PROCEDURE No. 6

1. Title: Size Estimation (NR and SLOC)

2. Prediction or Estimation Parameter Supported: Fault Density

3. Objectives: To determine fault density, some measure of size must be used as the sample size (denominator). Described here are two alternatives: number of system requirements and number of source lines of code.

4. Overview: During the early phases of a development, problems identified are typically at a system or subsystem level. In order to provide some relative measure of the significance of these problems, a sizing measure is needed at a system level. A simple measure of size is the number of functions required in the System Specification (the number of shall statements may accurately reflect this).

Later in the development, an estimate or actual count (during coding) of the number of lines of code will provide a basis for judging problems identified at the module level. Program size is not generated automatically by operating system software, since the number of printed lines may include comments, declarations. blank lines, or lines containing multiple statements. Our accepted definition of source lines of code will be the number of executable statements.

5. Assumptions/Constraints: It is assumed that the number of executable statements can be compared among systems. This assumption is not likely to hold when systems are written in different languages: a comparison of FORTRAN with LISP or APL would be misleading because of the greater compactness of LISP and APL in many applications. However, most of the HOLs to be considered are similar enough to make this metric sufficiently reliable for estimates of program size.

6. Limitations: Counting the number of requirements involves significant discipline. Use of a formal requirements specification language simplifies the task significantly. Use of the concept of function points is another alternative. The key is to be consistent.

7. Applicability: Estimates of program size should be available during all development phases.

8. Required Inputs: The Size Estimates will be based on the Requirements Specifications and the software.

9. Required Tools: Requirements Specification languages or analysis tools such as PSL/PSA, SREM, RTT, USE-IT are applicable. Compilers or code audit routines generally provide lines of code counts.

10. Data Collection Procedures: To determine the number of requirements, individual requirements must be itemized by analysis of the Requirements Specification. Data Collection Worksheets 3 in Appendix C can be used.

To estimate lines of code, use of senior personnel familiar with the specific application or reference to a historical data base which provides code counts for certain applications are the most proven techniques. Data Collection Worksheet 4 in Appendix C can be used.

Use of the compiler output or code auditors provide actual counts once coding is underway.

11. Outputs: Program size (SLOC) and number of requirements (NR) are reported.

12. Interpretation of Results: Program size can be used as a predictor of error rates. However, its primary use in this research is in combination with Software Discrepancy Reports in determining fault density.

13. Reporting: Program size is reported to the project manager as required for estimating resource requirements.

14. Forms: The standard worksheets in Appendix C provide for reporting Program Size.

15. Potential/Plans for Automation: Moderate revisions of existing system software should make it possible to obtain more accurate counts of program size in terms of number of lines of executable code.

16. Remarks: As noted, the measurement of program size has been used in the past as a predictor of software quality. Program size should be correlated with software failure rates, where appropriate, to determine the significance of this metric.

PROCEDURE NO. 7

1. Title: Fault Density

2. Prediction or Estimation Parameter Supported: Fault Density

3. Objectives: Fault Density represents a measure of the number of faults in a software system.

4. Overview: Fault Density may be used to provide a preliminary indication of software reliability. Because of the functional relationship between this metric and the Failure Rate, it provides an alternative measure of software reliability. Its major advantages are that it is fairly invariant and that it can be obtained from commonly available data.

5. Assumptions/Constraints: The predicted fault density will depend in part on the review and test procedures used to detect software faults. In any case, there is no guarantee that all faults have been found. Although it can be used to estimate failure rates, it cannot be directly combined with hardware reliability metrics.

6. Limitations: As noted, the Fault Density estimates may be affected by the review and testing procedures.

7. Applicability: This number is confirmed during the formal testing phases where faults are observed and discrepancy reports formally recorded. During early phases of the development, a fault density measure can be obtained by using the number of problem reports documented during reviews or the prediction methodology.

8. Required Inputs: Estimates of Fault Density are obtained from software discrepancy reports. The number of faults reported, divided by the number of lines of executable code (or number of requirements during early phases of development), gives the required metric. Reference is made to Data Collection procedures 6 and 12.

9. Required Tools: Accurate records of software faluts are essential for this metric. A data base management system to prepare summary reports would simplify record keeping and preparation of calculation of Fault Density.

10. Data Collection Procedures: A count of software faults is obtained through inspection of software discrepancy reports. The number of lines of executable code will also be required. Use of a discrepancy report such as that at Worksheet 5 in Appendix C is recommended.

11. Outputs: The predicted Fault Density (RP) is the primary output. In addition, estimates of failure rates, based on the transformations described in Task 100, will also be output.

12. Interpretation of Results: The Fault Density is used as a predictor for the Failure Rate, and thus should provide an important indicator of software reliability in advance of full-scale system tests. It also can be compared with a specified fault density as a requirement or with industry averages represented in Table TS101-1. It is also an indicator of individual components that are potentially high risk elements or unreliable components.

13. Reporting: This metric is reported, together with the estimates of failure rates, to support predictions and estimates of software reliability.

14. Forms: Metric Worksheet 5 in Appendix C.

15. Potential/Plans for Automation: The software discrepancy reports may be kept in standard formats for access through a data base management system. The system should be sufficiently powerful to provide counts of errors for each module and to calculate fault densities, if the module lengths are available.

16. Remarks: The fault density, because of its functional relationship to failure rates, will provide an estimate of software reliability during coding and early testing.

PROCEDURE NO. 8

1. Title: Language Type (SL)

2. Prediction or Estimation Parameter Supported: Software Chacteristics

3. Objectives: Categorizes language or languages used in software unit as assembly or higher order language (HOL).

4. Overview: In the Language Type metric, the system is categorized according to language. Language Type has been shown to have an effect on error rates.

5. Assumptions/Constraints: Because of the significant effect that language can have on software reliability, use of this metric will provide an early indication of expected failure rates.

During the requirements phase, language requirements may be tentatively indicated, particularly when a new system must interface with existing software or hardware.

During the specifications phase, detailed information concerning proportions of HOL and assembly code will normally become available.

Finally, during integration and test, it may become necessary to change the specified proportion of assembly code in order to meet space, time, or performance constraints.

6. Limitations: Accuracy of this metric will depend on the accuracy of estimates of lines of HOL and assembly language code during early phases of development. While detailed specifications will normally include an estimate of program size, this estimate must be revised during software development.

7. Applicability: This metric is obtained during the preliminary design phase to provide an early warning of potential effects of language selection. Because of the higher error rates encountered when assembly language programming is used, it may indicate a choice of HOL rather than assembly language.

More importantly, it can provide a measure of the cost, in terms of higher error rates, to be balanced against projected savings in time and space, for a proposed design.

8. Required Inputs: Information is extracted manually from requirements or specifications documentation. During implementation and test, more accurate measures of the number of lines of code will be available from compilers or automated program monitors.

9. Required Tools: Information is extracted manually from existing documentation during requirements and specifications phases. During implementation and test, information will be available from compiler output or code auditors.

10. Data Collection Procedures: Initial estimates of lines of code will be extracted from existing documentation. When this information is not available, the value of the metric will be set to 1.0. Counts of the number of lines of source code may be obtained from compilations of software units. Comments and blank lines should not be included in this total, and it may be necessary to exclude them manually.

11. Outputs: The following outputs are required from this procedure,

ALOC - The number of lines in assembly language

HLOC - The number of lines in HOL

SLOC = ALOC + HLOC - total number of executable lines of code (see Data Collection Procedure 6).

These are combined according to the following formula:

$$SL = 1.4 * ALOC/SLOC + ALCO/SLOC$$

12. Interpretation of Results: When combined with other metrics, SL will indicate the degree to which the predicted or estimated error rate will be increased because of the use of assembly language. This information, when compared with the expected increase in efficiency through the use of assembly language, can be used as a basis for a decision concerning implementation language.

13. Reporting: The value of SL will be reported and combined with other measures in obtaining a predicted failure rate.

14. Forms: Forms for reporting the number of lines of code, the proportion lines in each stated category, and the composite SL are in Appendix C, Worksheet 4.

15. Potential/Plans for Automation: Language Type will normally be specified in requirements and specifications, and must be obtained manually.

16. Remarks: As research progresses, it may become possible to make finer distinctions among languages, and among versions of the same language. For this reason, the specific implementation should be included in this report. That is, the name of the

language, the version, the operating system and version, and the processor name and type should be reported when this information is available.

PROCEDURE NO. 9

1. Title: Module Size (SM)

2. Prediction or Estimation Parameter Supported: Software Characteristics

3. Objectives: Structured programming studies and Government procurement documents have frequently prescribed limits on module size, on the basis of the belief that smaller modules were more easily understood, and would therefore, be less likely to contain logical errors. This metric provides an estimate of the effect of module size, based on the proportions of modules with number of lines of executable code as follows:

No. of Modules	
u	Less than 200
w	200 to 3,000
x	Over 3,000

4. Overview: Inspection of compiler reports, editors, or source code will provide module length. Lines of code are counted on the same basis as that used in the Program Size metric.

5. Assumptions/Constraints: Lines of code include executable instructions. Comments and blank lines are excluded. Declarations, data statements, common statements, and other non-executable statements are not included in the total line count. Where single statements extend over more than one printed line, only one line is counted. If more than one statement is included on a printed line, the number of statements is counted.

Assembly language lines are converted to HOL line equivalents by dividing by an appropriate expansion factor, and program size is reported in source code lines or equivalents.

6. Limitations: The precision of the reported Module Size may be affected by human factors, if the reporter is required to count lines visually, or to revise the figure reported by the compiler or editor. When the project is large enough to support it, an automatic line counter, which would produce consistent line counts, should be supplied.

7. Applicability: This metric will not be available until detailed program specifications have been written. Estimates of module size will normally be included in specifications.

8. Required Inputs: Specifications containing module size estimates may be used for early computation of this metric. As modules are completed, more accurate figures for size will become

available. For existing software, module size is normally contained in system documentation; otherwise, it may be obtained through inspection of the code.

9. Required Tools: The compiler or editor will provide counts of the total number of lines in each module. Additional software tools could be provided to count lines of executable code, excluding comments and blank lines.

10. Data Collection Procedures: Compiler or editor output is examined to determine sizes for each module. Where counts include comments or blank lines, these must be eliminated to obtain a net line count. Modules are then categorized as shown above, and a count is made of the number of modules in each category.

11. Outputs: Results are reported in terms of the raw counts of the number of modules in each category, together with the resulting metric SM.

12. Interpretation of Results: In general, it has been assumed that any large modules will increase the potential failure rate of a software system. Later experiments will test this assumption.

13. Reporting: The values of u, w, and x will be reported.

14. Forms: Metric worksheet 4 in Appendix C includes this metric.

15. Potential/Plans for Automation: Compilers and editors typically provide enough data to compute this metric. A fully automated system would give more accurate estimates of the number of executable statements.

16. Remarks: More sophisticated measures of modularity should be explored.

PROCEDURE NO. 10

1. Title: Complexity (SX)

2. Prediction or Estimation Parameter Supported: Software Characteristics

3. Objectives: The logical complexity of a software component relates the degree of difficulty that a human reader will have in comprehending the flow of control in the component. Complexity will, therefore, have an effect on software reliability by increasing the probability of human error at every phase of the software cycle, from initial requirements specification to maintenance of the completed system. This metric provides an objectively defined measure of the complexity of the software component for use in predicting and estimating reliability.

4. Overview: The metric may be obtained automatically, where complexity = number of branches in each module + 1. This is McCabe's cyclomatic complexity metric.

5. Assumptions/Constraints: Some analogue of the complexity measure might be obtained during early phases -- for example, through a count of the number of appearances of THEN and ELSE in a structured specification or by counting branches in a Program Design Language description of the design - but actual complexity can be measured only as code is produced, at software implementation.

6. Limitations: Another limitation may be found in the possible interaction of this metric with length - longer programs are likely to be more complex than shorter programs -- with the result that this metric simply duplicates measurements of length.

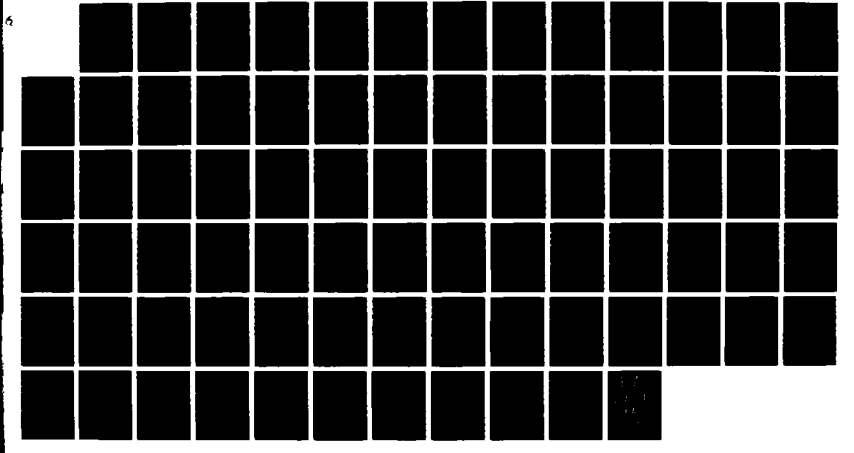
7. Applicability: Complexity measures are widely applicable across the entire software development cycle. Reliability models have not yet been defined for the Requirements phase, and probably cannot be applied unless a formalized specification language is used. To the extent that specifications are formalized, a complexity metric may be used. The measure to be used here, may be extracted automatically when code is produced. A series of measures will be taken and increases or decreases in complexity will be noted.

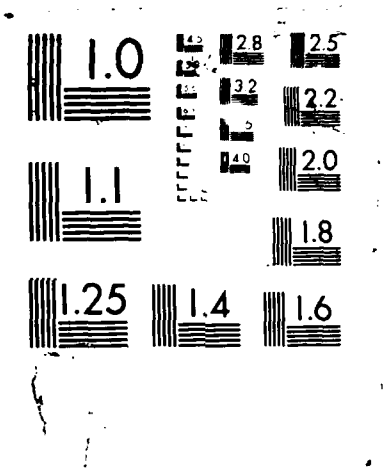
8. Required Inputs: Coded modules are the input for complexity measurement.

9. Required Tools: An analysis program which identifies and counting program branches [illegible]

10. Data Collection Procedure: [illegible] should be possible to initiate [illegible] filename of the code [illegible]

AD-A190 819 METHODOLOGY FOR SOFTWARE RELIABILITY PREDICTION VOLUME 2/2
2(U) SCIENCE APPLICATIONS INTERNATIONAL CORP SAN DIEGO
CA J MCCALL ET AL. NOV 87 RADC-TR-87-171-VOL-2
UNCLASSIFIED F30602-83-C-0118 F/G 12/5 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
4.0

a visual count of the number of edges or paths in a flowchart representation of the modules. Another approach would be to count the number of appearances of THEN, ELSE, GOTO, WHILE, and UNTIL together with a count of the number of branches following a CASE, computed GOTO, or Fortran IF statements.

11. Outputs: An complexity measure (SX) will be output.

12. Interpretation of Results: A large value for SX indicates a complex logical structure, which will affect the difficulty that a programmer will have in understanding the structure. This in turn will affect the reliability and maintainability of the software, since the probability of human error will be higher.

13. Reporting: Abnormally high values for SX should be reported to the program managers as an indication that the system is overly complex, and thus difficult to comprehend and error prone. Complex individual modules will also be identified by high values for sx(i).

14. Forms: The report form for each module and for the system as a whole should indicate the complexity, obtained either from an automated procedure or by hand. Metric worksheet 4 in Appendix C can be used.

15. Potential/Plans for Automation: A code auditor should be obtained or written to provide automated estimates of program complexity.

16. Remarks: Further experimentation with complexity metrics is desirable, and any automated tools written for this purpose should include alternative approaches, such as Halstead's metrics.

PROCEDURE NO. 11

1. Title: Standards Review (SR)

2. Prediction or Estimation Parameter Supported: Software Characteristics

3. Objectives: This metric represents standards compliance by the implementers. The code is reviewed for the following characteristics:

-- Design organized in top-down fashion,

-- Independence of module,

-- Module processing not dependent on prior processing,

-- Each module description includes input, output, processing, limitations,

-- Each module has a single entry and at most one routine and one exception exit.

-- Size of data base,

-- Compartmentalization of data base,

-- No duplicate functions, and

-- Minimum use of global data.

4. Overview: The purpose of this procedure is to obtain a score indicating the conformance of the software with good software engineering standards and practices.

5. Assumptions/Constraints: This data will be collected via QA reviews/walkthroughs of the code or audits of the Unit Development Folders or via a code auditor developed specifically to audit the code for standards enforcement.

6. Limitations: In general, components of this metric must be obtained manually and are thus subject to human error. However, the measures have been objectively defined and should produce reliable results. The cost of obtaining these measures, where they are not currently available automatically, may be high.

7. Applicability: This data is collected during the detailed design and more readily during the coding phase of a software development.

8. Required Inputs: Code

9. Required Tools: A code auditor can help in obtaining some of the data elements.

10. Data Collection Procedures: Use Metric Worksheet 11 in Appendix D and review (walkthrough) code.

11. Outputs: The number of modules problems with (PR) is identified.

12. Interpretation of Results: Noncompliance with standards not only means the code is probably complex, but it is symptomatic of an undisciplined development effort which will result in lower reliability.

13. Reporting: The modules which do not meet standards are reported via problem reports.

14. Forms: Metric worksheet 11 in Appendix D may be used.

15. Potential/Plans for Automation: In general, components of this procedure are inappropriate for automated collection. Implementation data can be collected automatically.

16. Remarks: Modification of the metric worksheets in Appendix D may be necessary to reflect different standards due to environment, application, or language.

PROCEDURE NO. 12

1. Title: Discrepancy Reports (DR)

2. Prediction or Estimation Parameter Supported: Fault Density and Failure Rate

3. Objectives: The basic metric for estimation will be the observed failure rate during testing. During Operation and Maintenance, the observed failure rate will also be used. The failure rate is based on the observed number of failures over time, which is derived from Discrepancy Reports and Execution Time measures.

4. Overview: A Software Discrepancy Report is generated at the time that an error is discovered or a failure occurs, typically during formal testing. An error is a discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. A failure occurs when the system or system component is unable to perform a required function within specified limits. A count of failures will be obtained from the Discrepancy Reports.

5. Assumptions/Constraints: Reported failure rates will not accurately reflect actual failure frequencies unless procedures for preparing and recording software problems are strictly enforced by project management. It is necessary to assume that differences in reported failure rates reflect actual differences between software components. Care must be taken to ensure that these differences are not merely artifacts of the collection procedures.

6. Limitations: Software induced failures will differ in seriousness, ranging from low-priority (easily corrected or avoided) to high-priority (results in mission failure). This information should appear on Discrepancy Reports, although it is not presently used directly in determining failure rates. The recommended categories of High, Medium and Low are defined in paragraph 8 below. Further research in the utilization of severity as a prediction criterion is warranted.

7. Applicability: Discrepancy Reports will be obtained during any formal reviews, coding and unit testing, CSC integration and testing, CSCI-level testing, acceptance testing, operational test and evaluation and O&M. MIL-STD 2167A references Software Problem Reports as backup to the Software Test Result Report. The Discrepancy Report described here meets that requirement as well as provides a mechanism for recording other discrepancies identified formally.

8. Required Inputs: Discrepancy Reports are documented by the program development staff, QA, customer testers or by the O&M staff as problems occur or are identified. Specific procedures

are to be included in the Software Quality Assurance Plan and Configuration Management Plan for the system. The Discrepancy Report contains an identification section in which a title and identification number are entered as well as as the author, date, and any references that should be included. It also is recommended that a discrepancy report includes a categorization scheme that will support trend analyses. The discrepancy report recommended in this guidebook (see Metric Worksheet 5) categorizes the discrepancy by type and criticality level. The criticality levels are:

High causes system to abort or fail to perform mission.

Medium incorrect results are obtained but does not necessarily jeopardize mission.

Low typically involves incorrect format, documentation errors, or miscalculations that does not threaten mission performance but should be fixed eventually.

Also described by the discrepancy report are the method used to detect the discrepancy, a description of the problem, the impact or effects of the problem, the recommended solution, and data on the test case and execution time, if the discrepancy was found during a test run. Discrepancy reports usually are approved after the appropriate fix has been made and QA releases it to configuration management for formal update of the current version of the software.

9. Required Tools: On-line entry of Discrepancy Reports will require storage of appropriate formats for the reports, and subsequent storage and retrieval facilities. Automatic computation of failure rates for system components is desirable.

10. Data Collection Procedures: Discrepancy Reports will be collected during system tests and operation as one of the responsibilities of the project manager.

11. Outputs: Discrepancy Reports will be accessible in a designated file. Their primary relevance will be in determination of the Failure Rate.

12. Interpretation of Results: Discrepancy Reports play a central role in the validation of software reliability metrics. The Failure Rate, based on information obtained from the reports, provides the baseline against which metrics for prediction and estimation are validated.

13. Reporting: For the purposes of this research, the Failure Rate will be reported. Since the Discrpancy Reports contain additional information of interest to project managers, they will be available for further reference and analysis.

14. Forms: A standard Discrepancy Report form is recommended (see

Data Collection Worksheet 5).

15. Potential/Plans for Automation: Discrepancy Reports are stored and retrieved through the file management system, but are prepared manually.

16. Remarks: Because of the importance of accurate and complete Discrepancy Reports in determination of failure rate, the collection and maintenance of the reports should be included in the management plan for any software included in the current reliability research project. Comparisons of failure rates between two systems will be misleading unless the same criteria have been used for both systems.

PROCEDURE NO. 13

1. Title: Execution Time (ET)

2. Prediction or Estimation Parameter Supported: Failure Rate

3. Objectives: Execution times are used in conjunction with Software Discrepancy Reports to obtain failure rates. The number of failures per time period is the basic reliability measure used in this Guidebook.

4. Overview: Execution time is the interval during which the central processing unit (CPU) of the computer executes the program. Two measures are suggeted. One is the actual CPU execution time. The other is computer operation time. Execution time will generally refer to each. Computer Operation time will be the default.

5. Assumptions/Constraints: Execution time cannot be directly compared between machines of different word length. Significant differences in machine architectures may make it impossible to compare execution times accurately. It is assumed, using the method given in item 10, that comparisons of sufficient accuracy can be made. The unit of time used in both measures of execution time is hours.

6. Limitations: The accuracy of execution time estimates will be affected by the type of timing device available in the system under test. Since not all operating systems are capable of sufficiently precise timing, statistical measures derived from the Execution Time measurements must not assume greater precision than is actually available.

7. Applicability: Execution time may be obtained during CSC integration and testing, CSCI-level testing, system integration and testing, operational test and evaluation and operations. Since it is used in conjunction with Software Discrepancy Reports, it should be obtained during the relevant reporting periods.

8. Required Inputs: Execution times are typically obtained from software operating system reports or test reports.

9. Required Tools: No special tools, other than those provided by the operating system, will be required.

10. Data Collection Procedures: Execution time is obtained from operating system records,or tester's logs, which typically report the execution time for each program or project on a run basis, as well as daily, weekly, or monthly totals. Where operating system reports are not available, execution time may be expressed in computer time, the time during which the computer (as contrasted

with the CPU) executes the program.

In cases in which execution time is not available, it may be estimated from total computer time with one of the following methods:

-- Running a benchmark HOL program on a mainframe on which execution time will be reported, and then running the same test case on the target computer.

-- Running a program on the target computer in a manner that will eliminate or minimize disk access (e.g., by putting data in memory) and output operations, thus obtaining essentially an execution time measurement, and then running the same test case in the normal manner.

-- Counting the number of I/O operations involved in a program and computing the nominal time for these from the computer instruction manual.

When comparisons are made between programs running on different computers, it is necessary to normalize execution time for word length and execution speed. Raw execution time is divided by the number of bits executed per second, which is obtained by multiplying computer word length in bits by the number of instructions per second. This figure may be modified in the case of machines which use more than one word for instructions, or which can have more than one instruction per word.

11. Outputs: Execution times are reported for use in calculation of failure rates. This guidebook recommends the use of computer operation time since it is more generally available.

12. Interpretation of Results: As noted in paragraph 10, raw execution times may be misleading, because of variations in computer word lengths, speed, and timing mechanisms employed. Because of the importance of the Failure Rate in validation of software reliability metrics, it will be essential to obtain accurate and reliable measures of Execution Time.

13. Reporting: Execution times are reported for use in Failure Rate measurements.

14. Forms: Worksheet 6 in Appendix C is prepared manually from data obtained from operating system outputs or tester's logs.

15. Potential/Plans for Automation: This metric is generated automatically by most operating systems.

16. Remarks: As noted, Execution Time cannot be compared directly between systems running on different machines. This problem can be expected to increase as specialized machine architectures are used (e.g., data base machines).

PROCEDURE NO. 14

1. Title: Failure Rate (F)

2. Prediction or Estimation Parameter Supported: Failure Rate

3. Objectives: The Failure Rate is the ultimate measure of software reliablity used in this guidebook. It represents the ground truth, which the metrics, in combination, are attempting to approximate. Failure Rate provides a measure of system reliability. Mission software failure probability is the product of software failure rate and mission duration; mission software reliability is 1 - mission software failure probability.

4. Overview: This metric is obtained by dividing the number of failures reported over a standard time period. Time-stamped software discrepancy reports are used to provide a count of system failures during the stated time period. The reports also typically indicate the module or CSCI with which they are associated.

5. Assumptions/Constraints: It is assumed that software discrepancy reports will provide an accurate measure of the failure rate of software over time. Preparation of discrepancy reports may not follow similar procedures on different projects. Even on the same project, as a deadline approaches, programmers may tend to feel that there is not time to prepare reports for failures that they perceive as minor, even though they might have prepared them at earlier times. The assumption, then, is that the programming environment is disciplined enough to enforce a consistent error reporting procedure. Automation of error reporting, if feasible, would help to increase consistency. Nevertheless, since the failure rate is essential for testing and validating all other metrics, it will be necessary to enforce consistency in the collection of data for this purpose.

6. Limitations: As noted in the preceding paragraph, consistency in data collection is assumed.

7. Applicability: This metric is obtained during operational tests and later operations and maintenance. It serves to validate predictions and estimates obtained during preceding phases of the software development cycle.

8. Required Inputs: Software discrepancy reports are used to measure the number of failures over time. The operating system is used to track the operation time.

9. Required Tools: None.

10. Data Collection Procedures: The software discrepancy reports are counted. In many cases, reports are maintained on disk, so that counts will be immediately available. Time stamps and modules are used to permit identification of reports with

designated time periods and software segments. The required metric is obtained by dividing the number of discrepancy reports by the number of hours of computer operation time to obtain the failure rate for any designated unit, CSCI,or system. The average failure rate during testing, F_{T1}, is calculated by taking the number of discrepancy reports recorded and dividing by the total amount of test time recorded. This average can be calculated anytime during testing and represents the current average failure rate. When calcualted, it is based on the current total number of discrepancy reports recorded and the current total amount of test operation time expended. It is expected that the failure rate will vary widely depending on when it is computed. For more consistent results, average failure rates should be calculated for each software test phase: CSC Integration and Testing, CSCI Testing; and, if required, for each system test phase: Systems Integration and Testing, and Operational Testing and Evaluation.

The failure rate at end of test, F_{T2}, is calculated by taking the number of discrepancy reports recorded during the last three test periods of CSCI Testing and dividing by the amount of test time recorded during these last three periods. This failure rate can be updated at the end of System Integration and test and at the end of Operational Test and Evaluation. A test period is defined as a test interval or session with specific test objectives. A test period could be a test run, a day or a month.

11. Outputs: The basic statistic output by this procedure is the failure rate. Since all metrics have been stated in terms of this rate, no further transformation should be required.

12. Interpretation of Results: The failure rate is interpreted as the primary measure of software reliability.

13. Reporting: Failure Rate is a basic measure of software quality and may be specified by the sponsoring agency or user. It is, therefore, essential to report failure rates to the project manager, to provide evidence that contractual requirements are being fulfilled.

14. Forms: Failure rates are to be reported for each module, CSCI, and system as part of the normal reporting procedure.

15. Potential/Plans for Automation: An automated procedure will provide an objective record of unit failures, although it is not likely that it will be able to provide complete information concerning the reasons for errors. In addition, it may not be able to detect failures in which outputs are not within required tolerances. In short, not all software failures are detectable by an automated system. Automation will be most valuable in maintaining error reports on-line, in an accessible form, for review by the project manager and quality control personnel.

16. Remarks: An accurate measure of failure rate is essential to the success of efforts to obtain appropriate metrics.

PROCEDURE NO. 15

1. Title: Test Effort (TE)

2. Prediction or Estimation Parameter Supported: Test Environment

3. Objectives: Test Effort is a measure of the quantity of testing to be performed. Three alternative measures are available. One is determined by the number of calendar days expended during each phase of testing, normalized by the total number of days (or hours) for the development effort. One is determined by the number of person days expended. One is determined by the amount of funds allocated to testing.

4. Overview: Estimates of the number of hours to be expended in testing are used during the early phases of the project. As actual numbers of hours become available, they are used to correct the early estimates.

5. Assumptions/Constraints: The Test Effort measurement requires access to labor hour data for a project and a work breakdown structure accounting system that delineates labor expended during testing. It is assumed that accurate figures for the hours of testing and total hours for the project are available. Because reported hours are not always accurate (e.g., because of unpaid overtime), some inaccuracy may appear in the reported hours.

6. Limitations: A measure of formal program testing may not include all testing performed. Typically, informal tests are performed at all levels throughout the development cycle. If these informal tests are frequent, and are not reported as such, the metric may be somewhat distorted, since the time for formal testing may be reduced without reducing the reliability of the software.

7. Applicability: Estimates of the amount of testing to be performed may be obtained throughout the software development cycle.

8. Required Inputs: For measurement of the amount of testing, job records from the software development project may be used. At earlier phases of the project, estimates of time to be spent in testing will be employed.

9. Required Tools: This factor will obtained manually from management reports.

10. Data Collection Procedures: Periodic project reports will be reviewed to obtain data concerning hours expended on software tests.

11. Outputs: The number of hours (or days) of testing, divided by the total number of hours (or days) in the development, will be used in computing this metric (AT). The value of the metric AT is used to determine the multiplier, TE.

12. Interpretation of Results: The amount of testing should provide an indication of software reliability, in that more thoroughly tested software is likely to contain fewer remaining errors. The effect of this measure could be balanced against the difficulty or complexity of the application, but no effective measure of the difficulty is available.

13. Reporting: A monthly report of the amount of testing would be appropriate, and would provide the project manager with a continuing record of effort expended on tests.

14. Forms: The metric worksheet 6 in Appendix C can be used.

15. Potential/Plans for Automation: This metric will be extracted manually from management reports.

16. Remarks: The project manager should keep accurate records of the time spent in software testing. In some instances, testing is not clearly broken out as a separate project task. Alternative methods for collecting this measure are by using funding instead of time.

PROCEDURE NO. 16

1. Title: Test Methodology (TM)

2. Prediction or Estimation Parameter Supported: Test Environment

3. Objectives: This metric is a measure or assessment of the test methodology. It is based on the techniques and tools employed.

4. Overview: The RADC Software Test Handbook, RADC TR 84-53, provides an approach for identifying what test tools and techniques should be employed based on the type and criticality of the software being developed. This simply uses that technique (or list of tools and techniques) as a score sheet to assess the test methodology actually employed.

5. Assumption/Constraints: Determination of this metric requires that the number of test tools be counted. This assumes that it is possible to count different tools and techniques meaningfully, although in some instances a tool may have several functions, or a number of tools may be integrated into a comprehensive testing environment. It also assumes that the distinction between software test tools and other support software (such as editors) is clear-cut.

6. Limitations: As noted, the use to test tools and techniques relies on several elements that may not be well-defined in particular applications.

7. Applicability: Information concerning projected use of test tools and techniques will be available at the requirements phase, as part of the Test Plan. The projected use of tools and techniques will be included in the Software Development Plan also. Reports on actual use of tools will become available during test and evaluation.

8. Required Inputs: Test Plans, Test Procedures, Software Development Plan.

9. Required Tools: Information concerning the use of tools and techniques will be obtained manually from project reports, as noted.

10. Data Collection Procedures: The Software Test Handbook, RADC TR 84-53, should be used to develop a list of tools and techniques that should be used. Each one used, confirmed by observing testing or reviewing documents, would be checked off.

11. Outputs: Output from this procedure will be the reported number of test tools and techniques were used (TU) and the total recommended (TT).

12. Interpretation of Results: The use of test tools and techniques is expected to produce a more effective and objective testing methodology, which should be reflected in greater system reliability.

13. Reporting: Information concerning the use of test tools and techniques should be reported back to the project monitor, as well as the project manager, to ensure that there is understanding of the role of tools in the software development effort.

14. Forms: Forms are available in the Software Test Handbook and in Data Collection Worksheet 7 in Appendix C.

15. Potential/Plans for Automation: This metric is essentially a description of test management, which is extracted manually from project documentation, rather than through the use of automated tools.

16. Remarks: Proper calibration of this metric, and procedures to avoid the effects of extreme values (such as zero), will be required.

PROCEDURE NO. 17

1. Title: Test Coverage (TC)

2. Prediction or Estimation Parameter Supported: Test Enviroment

3. Objectives: Test Coverage is a measure of the thoroughness of testing in terms of how thoroughly the code was executed during dynamic testing of the system.

4. Overview: Using available test tools, a count is taken which assesses coverage (VS). This coverage can be assessed during unit testing looking at paths executed, at integration testing looking at units and interfaces tested, or at system testing looking at requirements tested.

The following data will be obtained during the indicated test phases:

-- Unit Test

--- Percent of executable lines of code exercised during all unit tests

--- Percent of branches exercised during all unit tests

-- Integration and test

--- Percent of modules exercised during implementation and test

--- Percent of all interfaces exercised during implementation and test

-- Demonstration/Operational Test and Evaluation

--- Percent of functions exercised

--- Percent of user scenarios exercised

--- Percent of I/O options exercised

5. Assumptions/Constraints: Not all branches and calls are actually equal in determining software reliability. For example, a well-designed system may include a large number of error procedures which are never called during normal system operation. Some portions of code may be used only when hardware or software failures are encountered. It may be difficult to exercise these portions of code during system tests.

6. Limitations: It should be noted that the exercise of a

portion of code does not, in itself, provide any guarantee that the code will perform correctly over the full range of program variables. At best, it provides evidence that the code is capable of functioning for some value of the variables that it uses.

7. Applicability: This metric may be obtained during unit tests, integration and testing, and demonstration and operational test and evaluation.

8. Required Inputs: Test programs normally provide data concerning the extent of testing, as noted above. The following data elements will be required:

TP - Total number execution paths
PT - Number of execution paths tested
TI - Total number of inputs
IT - Number of inputs tested
NM - Total number of units
MT - Number of units tested
TC - Total number of interfaces
CT - Number of interfaces tested
NR - Total number of requirements
RT - Number of requirements tested

9. Required Tools: Appropriate test tools are available for exercising software systems and for obtaining required inputs. It will be necessary to identify appropriate test tools for specific systems to be tested.

10. Data Collection Procedures: The inputs described in paragraph 8 above are to be extracted during the project phases in paragraph 4. These are combined using the formula in Task 201 to obtain the Test Coverage metric, TC.

11. Outputs: The Test Coverage metric will be used in the computation of the Test Environment metric.

12. Interpretation of Results: A complete testing procedure would exercise all possible combinations of paths through the software system, using data for the full range of permissible and inpermissible (erroneous) values. Such tests of any reasonably complex system become expensive, because of the enormous number of combinations of values and paths to be exercised. The Test Coverage metric must therefore be interpreted in terms of the ultimate user of the system, the cost of failures, and mechanisms for recovery. From the point of view of cost-effectiveness, full tests of a system may not be preferable to less expensive partial tests, providing that the cost of failure is not excessive.

13. Reporting: Reports should indicate serious failures in the testing process, where tests have failed to cover significant portions of the software system. For the project manager, such reports are valuable.

14. Forms: Worksheet 8 in Appendix C is for each of the three

metrics.

15. Potential/Plans for Automation: Test Coverage can be computed automatically through the use of software test tools.

16. Remarks: Test Coverage is an important metric for evaluating the quality of testing that has been applied to the software system. It provides a measure of the degree of confidence that the manager can have in the results of testing.

PROCEDURE NO. 18

1. Title: Exception Frequency (EV)

2. Prediction or Estimation Parameter Supported: Operating Environment

3. Objectives: This metric represents the view that the greater the variability of inputs to the program, the more likely an unanticipated input will be encountered and the program will fail.

4. Overview: A measure of program variability (EV) will be obtained through a count of exception conditions that occur over a period of time. Hardware monitors will provide the required data. The value of EV may be represented as:

$$EV = .1 + 4.5EC$$

where EC is a count of the number of exceptions encountered in an hour.

5. Assumptions/Constraints: There has not been sufficient testing of variability as a possible factor in software system failures. In the form described here, however, it is plausible to suppose that the number of minor and recoverable problems, as measured by the number of exception conditions, is proportional to the number of major failures, and may be used during system implementation and test to estimate failure rate.

6. Limitations: This metric is derived from hardware and software exception reports, which are normally generated by the operating system. It will, therefore, provide the basis for estimating failure rates to be expected during operation of the system. It will not be available until initial system test.

7. Applicability: Exception Frequency is determined during the coding phase, testing, and O&M.

8. Required Inputs: Exception reporting is obtained from system monitors which generate records of hardware and software failures.

9. Required Tools: The appropriate system capabilities for monitoring, reporting, and summarizing exceptions must be available for use.

10. Data Collection Procedures: Records of hardware and software exceptions are obtained, from which a count of exceptions over a series of time periods will be prepared and averaged.

11. Outputs: The results are reported as EV, as defined above. This represents a normalized figure for the number of exceptions per time period. The normalization permits the value of EV to represent the degree of increase in expected failures, reflecting the frequency of exceptions.

12. Interpretation of Results: Hardware failures are likely to account for a substantial number of exceptions. A faulty disk or tape, or faulty components in the drive mechanisms for their supporting equipment, can generate large numbers of exceptions over a period of time. For this reason, it will be important to provide some explanation of the causes of the exceptions, to permit a proper interpretation of abnormally high exception rates.

13. Reporting: Exception frequencies should be reported back to the project monitor in cases in which excessive or anomalous values are encountered. This metric is valuable in estimating potential failure rates, by identifying specific modules or functions for which the anomaly rate is high.

14. Forms: Exception rates are based on data collected by hardware and software monitors and reported by operating system functions. This information can be entered into report forms to obtain the required value for EV.

15. Potential/Plans for Automation: A system for the collection of software metrics could include required functions for obtaining exception rates and for transforming them into the specified outputs.

16. Remarks: The exception rate appears to provide the basis for a highly accurate estimate of the failure rate to be expected during later system operations.

PROCEDURE NO. 19

1. Title: Workload (EW)

2. Prediction or Estimation Parameter Supported: Operating Environment

3. Objectives: The Workload metric represents an estimate of the workload of the system. It is thought to be more likely that a specific task will fail in a heavily loaded system than in a lightly loaded system.

4. Overview: One measure of workload is the amount of overhead being utilized. It represents how much I/O, system calls, swapping/paging,etc is going on. In most mainframes this measure is reported by the operating system. The EW is obtained by calculating the ratio of execution time to execution time minus overhead.

5. Assumptions/Constraints: To obtain a metric which will predict reliability, it will be necessary to obtain a figure for overhead which is typical of the times when there is significant activity. Overall averages for workload will have little predictive value if they include long periods when the computer system is completely idle. For that reason, the average should be computed during peak usage.

6. Limitations: Estimates of workload should accurately reflect conditions in the operating environment. The possiblity of rapid system degradation under conditions of heavy overload should be considered. Another point for consideration is possibility that system reliability will degrade -- i.e., the failure rate will increase -- in a non-linear fashion as the workload increases. There may be no failures attributable to system overload while the workload is less than, say, 95 percent; at this point, the failure rate begins to increase dramatically. The manner in which this metric is calculated assumes a linear relationship between workload and failure rate.

7. Applicability: A measure of workload can be determined during the coding phase, testing, and O&M. We are attempting to estimate what the workload will be like in operation. Stress tests, during which workload is deliberately kept at a high level, can be used to measure the effect.

8. Required Inputs: Computation of this metric will require data concerning total run time and overhead time.

9. Required Tools: Information required for this metric is normally available through the system monitor.

10. Data Collection Procedures: As data concerning overhead and total run time become available, the ratio is computed and

reported.

Often, the ratio appears in accounting information produced by the system monitor.

11. Outputs: The ratio (EW) is reported as an output to the computation of the Operating Environment metric.

12. Interpretation of Results: In general, it may be expected that system performance will degrade rapidly as the CPU approaches saturation. The problem for consideration will be the extent to which software degrades in a nondestructive manner, maintaining as many mission-critical functions as possible. Outputs from this metric should be useful in identifying a point at which degraded performance begins. Typically, Government specifications required that no more than 75 percent of system capacity be used, i.e., that there is a 25 percent margin for error, for mission-critical systems.

13. Reporting: Busy time or workload should be reported with other management data concerning resources use.

14. Forms: Workload is obtained from system management records, which are normally generated automatically by the operating system.

15. Potential/Plans for Automation: Information concerning workload and overhead is routinely gathered in automated computer management systems.

16. Remarks: It should be noted that this metric could also be the difference between idle time and total time. In a time-shared system, a significant portion of the busy time may be occupied by system overhead. In addition, in some applications, time that would otherwise be idle is absorbed by low-priority tasks (such as checking data bases for consistency) that would otherwise be idle. If a low-priority task is used to soak up idle time, it may produce a misleading estimate of the actually busy time -- i.e., the time used by higher-priority tasks.

APPENDIX C
METRIC DATA COLLECTION WORKSHEETS

Appendix C contains metric worksheets used to collect metric data during development phases. Nine different worksheets are applied to development products in different phases and at different levels of abstraction. These worksheets are modeled after those documented in RADC TR 85-37 listed here:

1. Metric Worksheet 0, system level, system/software requirements analysis.

2. Metric Worksheet 1, CSCI level, software requirements analysis.

3. Metric Worksheet 2, CSCI level, preliminary design.

4. Metric Worksheet 3A, CSCI level, detailed design.

5. Metric Worksheet 3B, unit level, detailed design.

6. Metric Worksheet 4A, CSCI level, code and unit testing.

7. Metric Worksheet 4B, unit level, code and unit testing.

One difference is that only those worksheet items pertinent to reliability prediction and estimation are included in this appendix. Any questions relating to definition, explanation, or application of these worksheets should be referred to RADC TR 85-37. Another difference is that the worksheets related to the Quality Review (SQ) metric and the Standard Review (SR) have been separated and reorganized in Appendix D.

Terminology used in the worksheets generally is consistent with DoD-STD-2167A (e.g., CSCI, unit). The term "software" is used in a broad sense and refers both to the end product (code, data and documentation) and to the product in its current stage of evolutionary development. A glossary is in [BOWE85]. An index of the worksheets showing the phases of development when they are applicable is provided in Table C-1. Multiple occurences of worksheets in Table C-1 represent either the application of a subset of the worksheet applicable to that phase or the fact that the worksheet could be updated at that phase if required.

TABLE C-1
METRIC WORKSHEET INDEX

PHASE	LEVEL	APPLICABLE METRIC	WORKSHEET
CONCEPT INITIATION	SYSTEM	APPLICATION TYPE	0
SYSTEM SOFTWARE DEFINITION	SYSTEM	APPLICATION TYPE	0
SOFTWARE REQUIREMENTS ANALYSIS	SYSTEM	APPLICATION TYPE DEVELOPMENT ENVIRNOMENT	0 1
SOFTWARE REQUIREMENTS ANALYSIS	CSCI	ANOMALY MGMT TRACEABILITY QUALITY REVIEW RESULTS	2 3 10 (Appendix D) 5
PRELIMINARY DESIGN	SYSTEM	APPLICATION TYPE DEVELOPMENT ENVIRONMENT	0 1
DETAILED DESIGN	CSCI	ANOMALY MGMT TRACEABILITY QUALITY REVIEW RESULTS	2 3 10 (Appendix D) 5
	CSCI	ANOMALY MGMT TRACEABILITY QUALITY REVIEW RESULTS	2 3 10 (Appendix D) 5
	UNIT	ANOMALY MGMT TRACEABILITY QUALITY REVIEW RESULTS	2 3 10 (Appendix D) 5
CODEING AND UNIT TESTING	SYSTEM	APPLICATION TYPE DEVELOPMENT ENVIRONMENT	0 1
CODING AND UNIT TESTING (CONT)	CSCI	ANOMALY MGMT TRACEABILITY SOFTWARE IMPLEMENTATION LANGUAGE TYPE MODULARITY COMPLEXITY STANDARDS REVIEW RESULTS	2 3 4 4 4 11 (Appendix D) 5
	UNIT	ANOMALY MGMT TRACEABILITY LANGUAGE TYPE COMPLEXITY STANDARDS REVIEW RESULTS	2 3 4 4 11 (Appendix D) 5
CSC INTEGRATION AND TESTING	SYSTEM	TEST EFFORT TEST METHODOLOGY TEST COVERAGE	6 7 8 5
OPERATIONAL TEST AND EVALUATION	SYSTEM	WORKLOAD VARIABILITY OF INPUT	9 9

METRIC WORKSHEET 0
SYSTEM SOFTWARE DEFINITION
SYSTEM LEVEL

GENERAL INFORMATION

1. PROJECT _____

2. DATE _____

3. ANALYST _____

4. PRODUCT _____

5. SOURCE DOCUMENTATION

_____

_____

_____

_____

_____

METRIC WORKSHEET O

APPLICATION TYPE (A)

Categorize software application according to application area as follows (Circle one category in each scheme):

SYSTEM NAME:_____ CSCI NAME(if Applicable)_____

APPLICATION (for System Level)	FUNCTION (for CSCI Level)
• AIRBORNE SYSTEMS - MANNED SPACECRAFT - UNMANNED SPACECRAFT - MIL-SPEC AVIONICS - COMMERICAL AVIONICS	• EVENT CONTROL • PROCESS CONTROL • PROCEDURE CONTROL
• STRATEGIC SYSTEMS - C^3I - STRATEGIC C^2 PROCESSING - INDICATIONS AND WARNING - COMMUNICATIONS PROCESSING	• MESSAGE PROCESSING • SENSOR AND SIGNAL • PATTERN AND IMAGE
• TACTICAL SYSTEMS - STRATEGIC C^2 PROCESSING - COMMUNICATION PROCESSING - TACTICAL C^2 - TACTICAL MIS - MOBILE - EW/ECCM	• EXECUTIVE/OPERATING SYSTEM • SUPPORT SOFTWARE/UTILITIES • RESOURCE MANAGEMENT/CONTROL
• PROCESS CONTROL SYSTEMS - INDUSTRIAL PROCESS CONTROL	• SCIENTIFIC/ANALYTICAL PROCESSING • DECISION AND PLANNING AIDS
• PRODUCTION SYSTEMS - MIS - DECISION AIDS - INVENTORY CONTROL - SCIENTIFIC	• DATA MANAGEMENT • DISTRIBUTION/COMMUNICATION • DISPLAY/DATA PRESENTATION
• DEVELOPMENTAL SYSTEMS - SOFTWARE DEVELOPMENT TOOLS - SIMULATION - TEST BEDS - TRAINING	• DIAGNOSTICS
TIMING CATEGORY	
• REAL TIME • ON-LINE	• BATCH • SUPPORT

METRIC WORKSHEET 1
DEVELOPMENT ENVIRONMENT (D)

Categorize development environment according to the following:

DEVELOPMENT ENVIRONMENT

( ) Organic Mode: The software team is part of the organization served by the program.

( ) Semi-detached Mode: The software team is experienced in the application but not affiliated with the user.

( ) Embedded Mode: Personnel operate within tight constraints. The team has much computer expertise, but is not necessarily very familiar with the application served by the program. System operates within strongly coupled complex of hardware, software, regulations, and operational procedures.

In addition, if data is available, complete the following supplementary checklist:

SUPPLEMENTARY CHECKLIST FOR DEVELOPMENT ENVIRONMENT

Identify all characteristics that are applicable to the development environment to be used on subject project:

ORGANIZATIONAL CONSIDERATIONS

( ) Separate design and coding
( ) Independent test organization
( ) Independent quality assurance
( ) Independent configuration management
( ) Independent verification and validation
( ) Programming team structure
( ) Educational level of team members above average
( ) Experience level of team members above average

METHODS USED

( ) Definition/Enforcement of standards
( ) Use of higher order language (HOL)
( ) Formal reviews (PDR, CDR, etc.)
( ) Frequent walkthroughs
( ) Top-down and structured approaches
( ) Unit development folders
( ) Software development library
( ) Formal change and error reporting
( ) Progress and status reporting

DOCUMENTATION

( ) System Requirements Specification
( ) Software Requirements Specification
( ) Interface Design Specification
( ) Software Design Specification
( ) Test Plans, Procedures and Reports
( ) Software Development Plan
( ) Software Quality Assurance Plan
( ) Software Configuration Management Plan
( ) Requirements Traceability Matrix
( ) Version Description Document
( ) Software Discrepancy Reports

TOOLS USED

( ) Requirements Specification Language
( ) Program Design Language
( ) Program Design Graphical Technique (flowchart, HIPO, etc.)
( ) Simulation/Emulation
( ) Configuration Management
( ) Code Auditor
( ) Data Flow Analyzer
( ) Programmer Workbench
( ) Measurement Tools
( ) Others

Count the number checked. Assign that number divided by 38 to D_C.

D_C = No. checked/38 = _____/38 = _____

METRIC WORKSHEET 2
ANOMALY MANAGEMENT

GENERAL INFORMATION

1. PROJECT _____

2. DATE _____

3. ANALYST _____

4. PRODUCT _____

5. SOURCE DOCUMENTATION

_____

_____

_____

_____

_____

6. PHASE: SRR _____
PDR _____
CDR _____
CODING _____

(Check applicable one)

7. LEVEL: System _____
CSCI _____ NAME _____
CSC _____ NAME _____
UNIT _____ NAME _____

(complete applicable level)

METRIC QUESTIONS

ANOMALY MANAGEMENT

The following checklists are used to assess the degree to which anomaly management (error tolerance) is being built into a software system. The checklists should be applied as follows:

CHECKLIST	APPLICATION
2B1	At Software Requirements Review
2B2	At Preliminary Design Review
2B3	At Detailed Design Review
2B4	At Unit level during coding

Note: Complete the worksheets as follows. Enter a value if required on the line next to the question in the value column. Check Yes or No on the line if question requires a yes or no resonse. Check NA to a question that is not applicable and these do not count in calculation of metric.

CHECKLIST 1B1 - SRR

	VALUE	YES	NO	NA
AM.1(1) a. How many instances are there of different processes (or functions, subfunctions) which are required to be executed at the same time (i.e., concurrent processing)?				
b. How many instances of concurrent processing are required to be centrally controlled?				
c. If b/a < 1, Circle N. If b/a = 1, Circle Y.				
AM.1(2)a. How many error conditions are required to be recognized (identified)?				
b. How many recognized error conditions require recovery or repair?				
c. If b/a < 1, circle N, If b/a = 1, cicle Y.				
AM.1(3) Is there a standard for handling recognized errors such that all error conditions are passed to the calling function or software element?				
AM.1(4) a. How many instances of the same process (or function, subfunction) being required to execute more than once for comparison purposes (e.g., polling of parallel or redundant processing results)?				
b. How many instances of parallel/redundant processing are required to be centrally controlled?				
c. If b/a < 1, Circle N. If b/a = 1 > Circle N				
AM.2(1) Are error tolerances specified for all applicable external input data (e.g., range of numerical values legal combinations of alphanumerical values)?				
AM.3(1) Are there requirements for detection of and/or recovery from all computational failures?				

		VALUE	YES	NO	N/A
AM.3(2)	Are there requirements to range test all critical (e.g., supporting a mission-critical function) loop and multiple transfer index parameters before use?				
AM.3(3)	Are there requirements to range test all critical (e.g., supporting a mission-critical function) subscript values before use?				
AM.3(4)	Are there requirements to check all critical output data (e.g., data supporting a mission critical system function) before final outputting?				
AM.4(1)	Are there requirements for recovery from all detected hardware faults (e.g., arithematic faults, power failure, clock interrupt)?				
AM.5(1)	Are there requirements for recovery from all I/O device errors?				
AM.6(1)	Are there requirements for recovery from all communication transmission errors?				
AM.7(1)	Are there requirements for recovery from all failures to communicate with other nodes or other systems?				
AM.7(2)	Are there requirements to periodically check adjacent nodes or interoperating systems for operational status?				
AM.7(3)	Are there requirements to provide a strategy for alternate routing of messages?				
RE.1(1)	Are there requirements to ensure communication paths to all remaining nodes/communication links in the event of a failure of one node/link?				
RE.1(2)	Are there requirements for maintaining the integrity of all data values following the occurrence of anomalous conditions?				

	VALUE	YES	NO	NA
RE.1(3) Are there requirements to enable all disconnected nodes to rejoin the network after recovery, such that the processing functions of the system are not interrupted?				
RE.1(4) Are there requirements to replicate all critical data in the CSCI at two or more distinct nodes?				
AM SCORE. Count the number of Y's checked. Count the number of N's checked. Calculate the number of N's divided by the number of N's and Y's. Assignment that value to AM.	TOTALS			
	AM =			

CHECKLIST 182 - ANOMALY MANAGEMENT (SAT) - PDR

		VALUE	YES	NO	N/A
AM.3(1)	Are there provisions for recovery from all computational failures?				
AM.4(1)	Are there provisions for recovery from all detected hardware faults (e.g., arithematic faults, power failure, clock interrupt)?				
AM.5(1)	Are there provisions for recovery from all I/O device errors?				
AM.6(1)	Are there provisions for recovery from all communication tran checking information (e.g. checksum, parity bit) computed and transmitted with all messages?				
AM.6(3)	Is error checking information computed and compared with all message receptions?				
AM.6(4)	Are transmission retries limited for all transmissions?				
AM.7(1)	Are there provisions for recovery from all failures to communicate with other nodes or other systems?				
AM.7(2)	Are there provisions to periodically check all adjacent nodes or interoperating systems for operational status?				
AM.7(3)	Are there provisions for alternate routing of messages?				
RE.1(1)	Do communication paths exist to all remaining nodes/links in the event of a failure of one node/link?				
RE.1(2)	Is the integrity of all data values maintained following the occurance of anomalous conditions?				
RE.1(3)	Can all disconnected nodes rejoin the network after recovery, such that the processing functions of the system are not interrupted?				
RE.1(4)	Are all critical data in the system (or CSCI) replicated at two or more distinct nodes, in accordance with specified requirements?				

VALUE	YES	NO	[illegible]
TOTALS			
AM =			

AM SCORE: Count the number of Y's circled and the number of N's circled. Calculate the ratio of the number of N's divided by the total number of N's and Y's. Assign that value to AM.

CHECKLIST 1B3 - ANOMALY MANAGEMENT (SA) - CDR

		VALUE	YES	NO	NA
AM.1(3)	a. How many units in CSCI? NM =				
	b. For how many units, when an error condition is detected, is resolution of the error determined by the calling unit?				
	c. Calculate b/a and enter score.				
	d. If b/NM 0 check N, otherwise check Y.				
AM.2(2)	Are values of all applicable external inputs with range specifications checked with respect to specified range prior to use?				
AM.2(3)	Are all applicable external inputs checked with respect to specified conflicting requests prior to use?				
AM.2(4)	Are all applicable external inputs checked with respect to specified illegal combinations prior to use?				
AM.2(5)	Are all applicable external inputs checked for reasonableness before processing begins?				
AM.2(6)	Are all detected errors, with respect to applicable external inputs, reported before processing begins?				
AM.2(7)	a. How many units, do not perform a check to determine that all data is available before processing begins.				
	b. Calculate a/NM and enter score.				
	c. If a/NM > 0 check N, otherwise check Y.				
AM.3(2)	Are critical loop and multiple transfer index parameters (e.g., supporting a mission-critical function) checked for out-of-range values before use?				
AM.3(3)	Are all critical subscripts (e.g., supporting a mission-critical function) checked for out-of-range values before use?				

	VALUE	YES	NO	NA
AM 3.4, Are all critical output data (e.g., supporting a mission-critical function) checked for reasonable values prior to final outputting?				
AM SCORE Count the number of Y's and N's circled and calculate the ratio of N's to the total number of N's and Y's. Assign that value to AM	TOTALS			
	AM =			

CHECKLIST 1B4 - ANOMALY MANAGEMENT (SA) - CODING

	VALUE	YES	NO	N/A
AM.1(3) When an error condition is detected, is resolution of the error determined by the calling unit?				
AM.2(7) Is check performed before processing begins to determine that all data is available?				
	TOTALS			

SCORE: Count the number of N's and divide by 2.

AM =

METRIC WORKSHEET 3
TRACEABLITITY ANALYSIS

TRACEABILITY

The following questions are used to assess the traceabilty of the system. Metric Worksheet 3 can be used to comply with these questions. The questions should be applied as follows:

QUESTIONS	APPLICATION
3A	At software Requirements Review
3B	At Preliminary Design Review
3C	At Detailed Design Review and During Coding

QUESTIONS 3A

TC.1(1) Is there a table(s) tracing all of the CSCI's allocated requirements to the parent system or the subsystem specification(s)?

ST SCORE: 1 if Y, 0 if N. SR = _____

QUESTIONS 3B

TC.1(1) Is there a table(s) tracing all the top-level CSC allocated requirements to the parent CSCI specification?

ST SCORE: 1 if Y, 0 if N. ST = _____

QUESTIONS 3C

TC.1(1) Does the description of each software unit identify all the specified requirements (at the top-level CSC or CSCI level) that the unit helps satisfy?

TC.1(2) Is the decomposition of top-level CSC's into lower-level CSC's and software units graphically depicted?

ST SCORE: 1 if both questions answered with Y.
0 if either or both answered with N. ST = _____

METRIC WORKSHEET 3

TRACEABILITY

Itemize individual requirements and trace their flowdown through design to code. Worksheet 3 is available to trace this requirements flowdown. Contractor specified format is acceptable.

SYSTEM REQUIREMENTS	DESIGN DERIVATIVE	SOFTWARE COMPONENT
<u>Example</u> SRS Para 2.4.1	SSS Para 2.4.1.1 SSS Para 2.4.1.2	PDS Para 3.10.1.1 PDS Para 3.10.1.2 PDS Para 3.10.1.3

Count Total Number of Itemized Requirements: NR=_____.

METRIC WORKSHEET 4
SIZE/COMPLEXITY/LANGUAGE DATA

Several of the measures used in the prediction methodology require sizing data about the software at various levels of detail. Such information as the overall size of the system and how it is decomposed into CSCI's, CSC's, and units, is required. Initially during a development, these data are estimates, then as the code is implemented, the actual size can be determined. Worksheet 4 can be used to record the data required by Data Collection Procedures 6, 8, 9 and 10. A worksheet should be filled out for each CSCI. Each unit's (MLOC) size and complexity (Sx(i)) is recorded in the right hand columns. An indication of the number of lines of higher order language (H) and assembly language (A) for each unit should be provided. The size data should be summed for all units in a CSC and for all CSC's in a CSCI. The totals are recorded at the bottom of the worksheet.

Complexity (SX(i)) is calculated as follows:

(1) Count the number of conditioned branch statements in a unit (eg. If, While, Repeat, DO/FOR LOOP, CASE). _____

(2) Count the number of conditioned branch statements in a unit (eg. GO TO, CALL, RETURN). _____

(3) Add (1) and (2) _____

WORKSHEET 4
SIZE/COMPLEXITY/LANGUAGE DATA

CSCI NAME =		DATE:		
CSC NAME	SLOC	UNIT NAME	MLOC	COMPLEXITY (sx(i))
CSCI SLOC=		TOTAL NO. OF UNITS: NM=		Σsx(i)=
TOTAL NO. OF HOL LOC: HLOC=		NO. OF UNITS < 200 LOC: U=		NO OF UNITS
TOTAL NO. OF AL LOC: ALOC=		NO. OF UNITS BETWEEN 200 AND 3000 LOC: W=		sx(i) ≥20: a=
		NO. OF UNITS > 3000 LOC: X=		20>Sx(i)>7: b=
				sx(i) <7: c=

WORKSHEET 5

During the Quality Review, Standards Review, or equivalent reviews such as Design and Code Inspections or Walkthroughs; during formal reviews such as SRR, PDR, CDR; and during testing, problems should be formally documented. The Discrepancy Report (Worksheet 5) or an equivalent problem report form should be used. The discrepancy report records the following information:

- Problem title and ID
- Analyst who uncovered problem
- Data it was found and phase of development
- Type of Problem
- Criticlity of Problem
- How it was detected
- Description of Problem
- What test run and how much test time was expended if it was found during testing
- Impact of Problem
- Solution
- Acknowledgement that it is a problem and date
- Acknowledgement that it has been fixed and dated

WORKSHEET 5 DISCREPANCY REPORT

PROBLEM TITLE: _____ PROBLEM NUMBER: _____

DATE: _____

PROGRAM ID: _____ ANALYST: _____

REFERENCES: _____

PROBLEM TYPE:

REQUIREMENTS	DESIGN	CODING		MAINTENANCE
• Incorrect Spec	• Requirements Compliance	• Requirements or Design Compliance	• Omitted Logic	• Incorrect Fix
• Conflicting Spec	• Choice of Algorithm	• Computation Implementation	• Interface	• Incompatible Fix
• Incomplete Spec	• Sequence of Operations	• Sequence of Operation	• Performance	
	• Data Definitions	• Data Definition		OTHER
	• Interface	• Data Handling		

CRITICALITY

HIGH _____ **MEDIUM** _____ **LOW** _____

METHOD DETECTION: _____

DESCRIPTION OF PROBLEM: _____

TEST EXECUTION: **TEST CASE ID:** **TEST EXECUTION TIME:**

EFFECTS OF PROBLEM:

RECOMMENDED SOLUTION:

APPROVED: _____ RELEASED BY: _____

DATE: _____ DATE: _____

WORKSHEET 6

TEST RECORD

During formal testing, it is important to record not only the problems encountered (see Worksheet 5) but also the amount of testing performed. This data allows calculation of the failure rate being experienced during testing. Worksheet 6 is provided to facilitate the required record keeping. Each individual Tester should complete these worksheets. Each individual test run should be recorded, the date it was run, a reference to a test plan or procedure if appropriate, reference to a descrepency report if there was a problem encountered during the test run, and the execution time of the test run. Note, successful test runs as well as unsuccessful test runs should be recorded with execution time. Reference to a discrepancy report is only made if a problem is encountered.

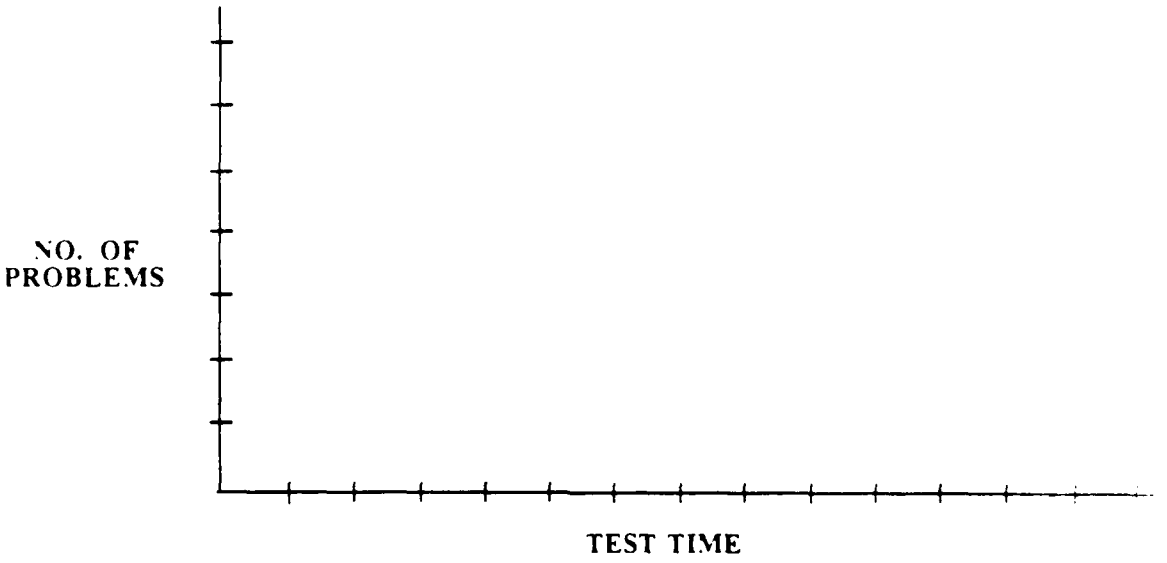
Execution time should be recorded in computer operation hours (wall clock time of run) and/or in CPU hours if CPU execution time is available. The measure of time should be indicated.

Worksheet 6A can be used to track testing progress. The units of time on the horizontal axis should be chosen to represent the test phase of the project. The number of problems recorded each test period should be plotted to facilitate observation of the trend in failure rate.

Worksheet 6A also supports calculation of the average failure rate during test test (F_{T1}) and the failure rate at end of test (F_{T2}).

Worksheet 6B is provided to support calculation of the Test Effort metric. The three alternative calculations are described in that worksheet. Inputs to these calculations come from management reports which track resource and budget expenditures and schedule.

WORKSHEET 6A
FAILURE RATE TREND



FAILURE RATE CALCULATION

AVERAGE FAILURE RATE DURING TEST:

$$F_{T1} = \frac{\text{Total number of Discrepancy Reports during Test}}{\text{Total Test Time}}$$

= _____ / _____ = _____

Failure Rate at end of Test:

$$F_{T2} = \frac{\text{No. of Discrepancy Reports during last 3 test periods}}{\text{Total Test Time during last 3 test periods}}$$

= _____ / _____ = _____

WORKSHEET 6B
TEST EFFORT

TEST EFFORT (TE)

Test effort represents the amount of effort applied to software testing. Three alternatives are available for evaluation of test effort. Each evaluates the percentage of effort, budget or schedule devoted to testing and compares that with a guideline of 40%. The recommended alternative, if data is available, is alternative 2. The second choice would be alternative 1, the last choice would be alternative 3.

ALTERNATIVE 1

DOLLARS

a. Budget in terms of dollars for the software testing effort. _____

b. Budget for the entire software development effort in terms of dollars _____

c. Calculate a/b and enter score _____

d. Calculate .40/c and enter score TM = _____

ALTERNATIVE 2

LABOR HOURS

a. Budget in terms of labor hours for the software testing effort _____

b. Budget for the entire software development effort in terms of labor hours _____

c. Calculate a/b and enter score _____

d. Calculate .40/c and enter score TM = _____

ALTERNATIVE 3

SCHEDULE

a. Schedule for software testing in terms of work days _____

b. Schedule for entire software development in terms of work days _____

c. Calculate a/b and enter score. _____

d. Calculate .40/c and enter score. TM = _____

METRIC WORKSHEET 7
TEST METHODOLOGY

RADC TR 84-53 provides procedures for identifying the appropriate test techniques and tools to use during a software development project. Worksheet 7A is reproduced from that report. It is the Selection Worksheet used in the report to identify the testing techniques recommended. The procedures in RADC TR 84-53 should be followed and the recommended techniques and tools should be documented in Worksheet 7B. Then, during testing, Worksheet 7B can be used as a checklist to assess which techniques and tools are actually used to test the software. The Test Plan, Specifications and Procedures as well as the Software Development Plan should be reviewed also.

The Test Methodology Metric, TM, then is based on the ratio of the applied techniques and tools, (TU) to the recommended techniques and tools (TT).

WORKSHEET 7A
SELECTION WORKSHEET FOR PATH 1

SOFTWARE TO BE TESTED_____

TCL_____ SOFTWARE CATEGORY_____

	SOFTWARE TEST TECHNIQUES	PATH 1	PATH 2	PATH 3	NOTES COMMENTS
STATIC ANALYSIS	Code Reviews				
	Error/Anomaly Detection				
	Structure Analysis/Documentation				
	Program Quality Analysis				
	Input Space Partitioning				
	A. Path Analysis				
	B. Domain Testing				
	C. Partition Analysis				
	Data-Flow Guided Testing				
DYNAMIC ANALYSIS	Instrumentation Based Testing				
	A. Path/Structural Analysis				
	B. Performance Measurement				
	C. Assertion Checking				
	D. Debug Aids				
	Random Testing				
	Functional Testing				
	Mutation Testing				
	Real-Time Testing				
	SYMBOLIC TESTING				
	FORMAL ANALYSIS				

WORKSHEET 7
TEST METHODOLOGY CHECKLIST

LIST TECHNIQUES AND TOOLS RECOMMENDED IN RADC TR 84-53	CHECK THOSE THAT ARE ACTUALLY USED
TOTAL NUMBER RECOMMENDED: TT=	TOTAL NUMBER USED: TU=

METRIC WORKSHEET 8
TEST COVERAGE

Collecting data to assess how thoroughly a software system is tested is difficult unless:

(1) A Requirements/Test Matrix has been developed

(2) A tool is used during testing which instruments the code and reports coverage data based on test case execution.

This worksheet assumes one or both these data sources are available in which case the following data can be collected:

DURING UNIT TEST (FOR EACH CSC OR UNIT):

TOTAL NUMBER OF EXECUTION PATHS:	TP=_____
TOTAL NUMBER OF EXECUTION PATHS TESTED:	PI=_____
TOTAL NUMBER OF INPUTS:	TI=_____
TOTAL NUMBER OF INPUTS TESTED:	IT=_____

DURING INTEGRATION TESTING (FOR EACH CSCI):

TOTAL NUMBER OF UNITS:	NM=_____
TOTAL NUMBER OF UNITS TESTED:	TM=_____
TOTAL NUMBER OF INTERFACES:	TC=_____
TOTAL NUMBER OF INTERFACES TESTED:	CT=_____

DURING SYSTEM TESTING:

TOTAL NUMBER OF REQUIREMENTS:	NR=_____
TOTAL NUMBER OF REQUIREMENTS TESTED:	RT=_____

Values for NM and NR were collected on other worksheets. Data Collected during unit testing can also be collected during integration testing and system testing. The unit and CSC level data should be accummulated and averaged at the CSCI level. Data collected at the Integration Test level can be collected during System Test also.

WORKSHEET 9
OPERATING ENVIRONMENT DATA

Three data items are required to derive the two metrics used to estimate the impact the operational environment will have on the failure rate. These data items; the amount of system overhead, the amount of execution time, and the number of execption conditions encountered during an hour of operating time, can be derived from the test environment, estimated, or calculated from a benchmark. In the first case, the data can be collected from the test environment and, based on the assumption that the test environment is representative of the operational environment, used for the metric calculation. In the second case, sample data can be collected from the test environment and based on an experienced analyst's judgement, that data can be adjusted to represent the relative workload and stress differences expected between the test environment and operational environment. In the third case, a benchmark can be run in the operational environment to provide the data. The data required is typically available from mainframe vendor operating system utilities. It is more difficult to collect in an embedded computer application where the target computer may be a special processor without a significant operating system capability.

To collect the data, monitor the processing during a specified time period (test period). This time period should be representative, or as close as possible, of the operational environment. During that time period collect the following:

Total Execution Time ET-_____

Amount of Operating System Overhead time: OS-_____

Number of exception conditions encountered: NEC - _____

Number of exception conditions encountered per hour of execution then is

EC - NEC/ET-_____

APPENDIX D
QUALITY REVIEW AND STANDARDS REVIEW WORKSHEETS

Appendix D contains worksheets used to conduct design and code reviews. These worksheets are recommended for use in conjunction with the software reliability prediction and estimation methodology. Alternative techniques that can be used are design and code inspections or design and code walkthroughs. The intent of these worksheets and these alternative techniques are to uncover discrepancies that should be corrected.

The worksheets contained in this Appendix relate to the metric worksheets in RADC TR 85-37 for metrics completeness, consistency, accuracy, autonomy, modular design and code simplicity.

The following checklists are used to assess the quality of the requirements and design representation of the software. Check the answer, yes, no or not applicable, or fill in the value requested in the appropriate column. The checklists should be applied as follows:

CHECKLIST	APPLICATION
10A	At Software Requirements Review
10B	At Preliminary Design Review
10C	At Detailed Design Review (CSCI Level)
10D	At Detailed Design Review (Unit Level)

METRIC WORKSHEET 10
QUALITY REVIEW

GENERAL INFORMATION

1. Project _____

2. Date _____

3. Analyst _____

4. Product _____

5. Source Documentation

_____

_____

_____

_____

CHECKLIST 10A - SRR QUALITY REVIEW

QUALITY REVIEW RESULTS (SRR). SYSTEM NAME:		VALUE	YES	NO	N/A
AC.1(3)	Are there quantitative accuracy requirements for all applicable inputs associated with each applicable function (e.g., mission-critical functions)?				
AC.1(4)	Are there quantitative accuracy requirements for all applicable outputs associated with each applicable function (e.g., mission-critical functions)?				
AC.1(5)	Are there quantitative accuracy requirements for all applicable constants associated with each applicable function (e.g., mission-critical functions)?				
AC.1(6)	Do the existing math library routines which are planned for use provide enough precision to support accuracy objectives?				
AD.1(1)	Are all processes and functions partitioned to be logically complete and self contained so as to minimize interface complexity?				
AD.2(1)	Are there requirements for each operational CPU/System to have a separate power source?				
AD.2(2)	Are there requirements for the executive software to perform testing of its own operation and of the communication links, memory devices, and peripheral devices?				
CP.1(1)	Are all inputs, processing, and outputs clearly and precisely defined?				
CP.1(2)	a. How many data references are identified?				
	b. How many identified data references are documented with regard to source, meaning, and format?				
	c. Calculate b/a and enter score.				
	d. If b/a . 1 check N, otherwise check Y.				
CP.1(3)	a. How many data items are defined (i.e., documented with regard to source, meaning, and format)?				
	b How many data items are referenced?				

		VALUE	YES	NO	NA
	c. if b/a < 1 circle N if b/a = 1 circle Y				
CP.1(5)	Have all defined functions (i.e., documented with regard to source, meaning, and format) been referenced?				
CP.1(6)	Have all system functions allocated to this CSCI been allocated to software functions within this CSCI?				
CP.1(7)	Have all referenced functions been defined (i.e., documented with precise inputs, processing, and output requirements)?				
CP.1(8)	Is the flow of processing (algorithms) and all decision points (conditions and alternate paths) in the flow described for all functions?				
CS.1(1)	Have specific standards been established for design representations (e.g., HIPO charts, program design language, flow charts, data flow diagrams)?				
CS.1(2)	Have specific standards been established for calling sequence protocol between software units?				
CS.1(3)	Have specific standards been established for external I/O protocol and format for all software units?				
CS.1(4)	Have specific standards been established for error handling for all software units?				
CS.1(5)	Do all references to the same CSCI function use a single, unique name?				
CS.2(1)	Have specific standards been established for all data representation in the design?				
CS.2(2)	Have specific standards been established for the naming of all data?				
CS.2(3)	Have specific standards been established for the definition and use of global variables?				
CS.2(4)	Are there procedures for establishing consistency and concurrency of multiple copies (e.g., copies at different nodes) of the same software or data base version?				
		TOTALS			
		DR =			

	VALUE	YES	NO	N/A
CS.2(5) Are there procedures for verifying consistency and concurrency of multiple copies (e.g., copies at different nodes) of the same software or data base version?				
CS.2(6) Do all references to the same data use a single, unique name?				
SQ INPUT: Count all N's Assign number to DR.	TOTALS			
	DR =			

CHECKLIST 10B - PDR QUALITY REVIEW

	VALUE	YES	NO	NA
QUALITY REVIEW RESULTS (SQ) CSCI NAME:_____				
AC.1(7) Do the numerical techniques used in implementing applicable functions (e.g., mission-critical functions) provide enough precision to support accuracy objectives?				
AU.1(1) Are all processes and functions partitioned to be logically complete and self-contained so as to minimize interface complexity?				
AU.1(4) a. How much estimated process time is typically spent executing the entire CSCI?				
b. How much estimated processing time is typically spent in execution of hardware and device interface protocol?				
c. if b/(b+a) > .3, circle N if b/(b+a) < .3, circle Y				
AU.2(2) Does the executive software perform testing of its own operation and of the communication links, memory devices, and peripheral devices?				
CP.1(1) Are all inputs, processing, and outputs clearly and precisely defined?				
CP.1(2) a. How many data references are defined?				
b. How many identified data references are documented with regard to source, meaning, and format?				
c. if < 1, circle N if b/a = 1, circle Y.				
CP.1(3) a. How many data items are defined (i.e., documented with regard to source, meaning, and format)?				
b. How many data items are referenced?				
c. Calculate b/a and enter score.				
d. If b/a < 1 check N, otherwise check Y				

		VALUE	YES	NO	NA
CP.1(4)	a. How data references are identified? b. How many identified data references are computed or obtained from an external source (e.g., referencing global data with preassigned values, input parameters with preassigned values)? c. if b/a < 1, circle N if b/a = 1, circle Y				
CP.1(6)	Have all functions for this CSCI been allocated to top-level CSC's of this CSCI?				
CP.1(9)	Are all conditions and alternative processing options defined for each decision point?				
CP.1(11)	a. How many software discrepancy reports have been recorded, to date? b. How many recorded software problem reports have been closed (resolved), to date? c. Calculate b/a and enter score. d. If b/a < .75 circle N, otherwise circle Y				
CS.1(1)	Are the design representations in the formats of the established standard?				
CS.1(5)	Do all references to the same top-level CSC use a single, unique name?				
CS.2(1)	Does all data representation comply with the established standard?				
CS.2(2)	Does the naming of all data comply with the established standard?				
CS.2(3)	Is the definition and use of all global variables in accordance with the established standard?				
CS.2(4)	Are there procedures for establishing consistency and concurrency of multiple copies (e.g., copies at different nodes) of the same software or data base version?				

	VALUE	YES	NO
CS.2(5) Are there procedures for verifying consistency and concurrency of multiple copies (e.g., copies at different nodes) of the same software or data base version?			
CS.2(6) Do all references to the same data use a single, unique name?			
SQ INPUT: Count all N's. Assign to DR.	TOTALS		
	DR =		

CHECKLIST 10C - CDR QUALITY REVIEW (CSCI)

		VALUE	YES	NO	NA
	QUALITY REVIEW RESULTS (SQ) CSCI NAME:_____				
AU.1(2)	a. How many estimated executable lines of source code? (total from all units) LOC-				
	b. How many estimated executable lines of source code necessary to handle hardware and device interface protocol?				
	c. if b/LOC > .3, circle N. if b/LOC ≤ .3, circle Y.				
AU.1(3)	a. How many units in CSCI? NM-				
	b. How many units perform processing of hardware and/or device interface protocol?				
	c. If b/(NM) > .3, circle If b/(NM) ≤ .3, circle				
AU.1(4)	a. How much estimated processing time is typically spent executing the entire CSCI?				
	b. How much estimated processing time is typically spent in execution of hardware and device interface protocol units?				
	c. if b/(a) > .3, circle N if b/(a) ≤ .3, circle Y				
CP.1(1)	a. How many units clearly and precisely define all inputs, processing, and outputs?				
	b. if b/(NM) > .3, circle N if b/(NM) < .3, circle Y				
CP.1(2)	a. How many data references are identified? (total from all units)				
	b. How many identified data references are documented with regard to source, meaning, and format? (total from all units)				
	c. Calculate c/b and enter value (total from all units)				

		VALUE	YES	NO	NA
	d. Calculate c/b and enter score.				
	e. If d > 0 check N, otherwise check Y.				
CP.1(3)	a. How many data items are defined (i.e., documented with regard to source, meaning, and format)?				
	b. How many data items are referenced?				
	c. Calculate b/a and enter				
	d. How many data references are computed or obtained from an external source (e.g., referencing global data with preassigned values, input parameters with preassigned values)? (total from all units)				
	e. Calculate d/b and enter score.				
	f. if c > 0 check N, otherwise check Y				
	g. if d > 0 check N, otherwise check Y				
CP.1(9)	a. How many units define all conditions and alternative processing options for each decision point? (total from all units)				
	b. Calculate a/NM and enter score.				
	c. If a/NM < 1 check N, otherwise				
CP.1(10)	a. For how many units, are all parameters in the argument list used?				
	b. Calculate a/NM and enter score.				
	c. If b < 1 check N, otherwise check Y				
CP.1(11)	a. How many software problem reports have been recorded, to date?				
	b. How many recorded software problem reports have been closed (resolved), to date?				
	c. Calculate b/a and enter score.				
	d. If c < .75 check N, otherwise check Y				

		VALUE	YES	NO	NA
CS.1(1)	a. For how many units are all design representations in the formats of the established standards?				
	b. Calculate a/NM and enter score.				
	c. If a/NM < 1 check N, otherwise check Y.				
CS 1(2)	a. For how many units does the calling sequence protocol (between units) comply with the established standard?				
	b. Calculate a/NM and enter score.				
	c. If a/NM < 1 check N, otherwise check Y				
CS.1(3)	a. For how many units does the I/O protocol and format comply with the established standard?				
	b. Calculate a/NM and enter score.				
	c. If a/NM < 1 check N, otherwise check Y				
CS 1(4)	a. For how many units does the handling of errors comply with the established standard?				
	b. Calculate a/NM and enter score.				
	c. If a/NM < check N, otherwise check Y				
CS.1(5)	a. For how many units do all references to the unit use the same, unique name?				
	b. Calculate a/NM and enter score.				
	c. If a/NM < 1, check N, otherwise check Y				
CS 2(1)	a. For how many units does all data representation comply with the established standard?				
	b. Calculate a/NM and enter score.				
	c. If a/NM < 1 check N, otherwise check Y				
CS.2(2)	a . For how many units does the naming of all data comply with the established standard?				
	b. Calculate a/NM and enter score.				
	c. If a/NM < 1, check N, otherwise check Y				

		VALUE	YES	NO	N/A
CS 2(3)	a. For how many units is the definition and use of all global variables in accordance with the established standard?				
	b. Calculate a/NM and enter score.				
	c. If a/NM < 1 check N, otherwise check Y				
CS 2(6)	a. For how many units do all references to the same data use a single, unique name?				
	b. Calculate a/NM and enter score.				
	c. If a/NM < 1 check N, otherwise check Y				
SQ SCORE	Total the number of No s. Assign to DR	TOTALS			

DR =

CHECKLIST 100 - CDR QUALITY REVIEW (UNIT)

	VALUE	YES	NO	NA
QUALITY REVIEW RESULTS				
UNIT NAME: _____				
CP.1(1) Are all inputs, processing, and outputs clearly and precisely defined?				
CP.1(2) a. How many data references are identified?				
b. How many identified data references are documented with regard to source, meaning, and format?				
c. If b/a < 1, circle N, otherwise circle Y?				
CP.1(4) a. How many identified data references are computed are computed or obtained from an external soruce (e.g., referencing global data with preassigned values, input parameters with preassigned values)?				
b. If b/DRI < 1, circle N, otherwise circle Y				
CP.1(9) Are all conditions and alternative processing options defined for each decision point?				
CP.1(10) Are all conditions and alternative processing options defined for each decision point?				
CP.1(1) Are all design representations in the formats of the established standards?				
CS.1(1) Are all design representations in the formats of the established standard?				
CS.1(2) Does the calling sequence protocol (between units) comply with established standard?				
CS.1(3) Does the I/O protocol and format comply with the established standard?				
CS.1(4) Does the handling of errors comply with the established standard?				
CS.1(5) Do all references to this unit use the same, unique name?				

	VALUE	YES	NO	N/
CS.2(2) Does the naming of all data comply with the established standard?				
CS.2(3) Is the definition and use of all global variables in accordance with the established standard?				
CS.2(6) Do all references to the same data use a single, unique name?				
	TOTALS			

SQ INPUT Count all No answers. Assign to DR.

DR =

METRIC WORKSHEET 11
STANDARDS REVIEW

GENERAL INFORMATION

1. PROJECT _____

2. DATE _____

3. ANALYST _____

4. PRODUCT _____

5. SOURCE DOCUMENTATION

_____

_____

_____

_____

The standards Review worksheet can be applied at the CSCI, CSC, or Unit level. Worksheet 11A applies to CSCI or CSC levels Worksheet 11B applies to Unit level.

WORKSHEET 11A STANDARDS REVIEW RESULTS (SR)

		VALUE	YES	NO	NA
	CSCI OR CSC NAME_____				
MO 1(2)	Are all units coded and tested according to structural techniques?				
MO 1(3)	a. How many units in CSCI or CSC?				
	b. How many units with estimated executable lines of soruce code less than 100 lines?				
	c. Calculate b/NM and enter score.				
MO 1(4)	a. How many parameters are there in the calling sequence? (total from all units)				
	b. How many calling sequence parameters are control variables (e.g., select an operating mode or submode, direct the sequential flow, directly influence the function of the software)? (total from all units).				
	c. Calculate b/a and enter score.				
	d. Calculate c/NM and enter score.				
MO 1(5)	a. For how many units is all input data passed into the unit through calling sequence parameters (i.e., no data is input through global areas or input statements?				
	b. Calculate a/NM and enter score				
MO 1(6)	a. For how many units is output data passed back to the calling unit (e.g., through calling sequence parameters of global areas)?				
	b. Calculate a/NM and enter score				
MO 1(7)	a. For how many units is control always returned to the calling unit when execution is completed?				
	b. Calculate a/NM and enter score				

		VALUE	YES	NO	NA
MO.1(8)	a. For how many units is temporary storage (i.e., workspace reserved for immediate or partial results) used only by the unit during execution (i.e., is not stored with other units)?				
	b. Calculate a/NM and enter score				
MO.2(2)	a. How many interfaces among software units?				
	b. How many unit interfaces include:				
	b1. content coupling				
	b2. common coupling				
	b3. external coupling				
	c. Calculate 1-(b1+b2+b3)(3*a) and enter score.				
MO.2(3)	a. How many unit interfaces include:				
	a1. control coupling				
	a2. stamp coupling				
	a3. data coupling				
	b. Caluclate ((a1+a2)/(2*TI) + (a3/TI)) and enter score				
MO.2(5)	What is the cohesion value?				
SI.1(2)	a. How many units are independent of the source of the input and the destination of the output?				
	b. Calculate a/NM and enter score.				
SI.1(3)	a. How many units are independent of knowledge of prior processing?				
	b. Calculate a/NM and enter score				
SI.1(4)	a. For how many units does the unit description/prologue include input,output processing, and limitations?				
	b. Calculate a/NM and enter score				

		VALUE	YES	NO	NA
SI.1.5	a. How many entrances? (total from all units)				
	b. How many exits? (total from all units)				
	c. Calculate (1/a+1/b)*1/2 and enter score				
	d. Calculate c/NM and enter score				
SI.1(6)	a. How many unique data items are in common block?				
	b. Calculate c/NM and enter score				
SI.1(7)	a. How many unique data items are in common blocks?				
	b. How many unique common blocks?				
	c. Calculate b/a and enter score				
SI.1(10)	Do the descriptions of all units identify all interfacing units and all interfacing hardware?				
SI.2(1)	a. How many units are implemented in a structural language or using a preprocessor?				
	b. Calculate a/NM and enter score				
SI.4(1)	a. For how many units is the flow of control from top to bottom (i.e., flow of control does not jump erratically)?				
	b. Calculate a/NM and enter score				
SI.4(2)	a. How many executable lines of code in this CSCI?				
	b. How many negative boolean and compound boolean expressions are used? (total from all units)				
	c. Calculate 1-(b/SLOC) and enter score				
	d. Calculate c/NM and enter score				
SI.4(3)	a. How many loops (e.g., WHILE, DO/FOR, REPEAT)? (total from all units)				

		VALUE	YES	NO	NA
	b. How many loops with unnatural exits (e.g., jumps out of loop, return statement)? (total from all units)				
	c. Calculate 1-(b/a) and enter score				
	d. Calculate c/NM and enter score				
SI.4(4)	a. How many iteration loops (i.e. DO/FOR loops)? (total from all units)				
	b. How many iteration loops are indices modified to alter fundamental processing of the loop? (total from all units)				
	c. Calculate 1-(b/a) and enter score				
	d. Calculate c/NM and enter score				
SI.4(5)	a. How many units are free from all self-modification of code (i.e., does not alter instructions, overlays of code, ect.)?				
	b. Calculate a/NM and enter score				
SI.4(6)	a. How many statement labels, excluding labels for format statements? (total from all units)				
	b. Calculate 1-(a/SLOC) and enter score				
	c. Calculate b/NM and enter score				
SI.4(7)	a. What is the maximum nesting level? (total from all units)				
	b. Calculate 1/a and enter score				
	c. Calculate b/NM and enter score				
SI.4(8)	a. How many branches, conditional and unconditional? (total from all units)				
	b. Calculate b/NM and enter score				
SI.4(9)	a. How many declaration statements? (total from all units)				
	b. How many manipulation statements? (total from all units)				
	c. Calculte 1((a+b)/SLOC) and enter score				
	d. Calculate c NM and enter score				

		VALUE	YES	NO	NA
SI.4(10)	a. How many total data items, local and global, are used? (total from all units) DD				
	b. How many data items are used locally (e.g., variables declared locally and value parameters? (total from all units)				
	c. Calculate b/a and enter score.				
	d. Calculate c/NM and enter score.				
SI.4(11)	a. Calculate DD/SLOC and enter score.				
	b. Calculate A/NM and enter score.				
SI.4(12)	a. How many units, does each data item have a single use (e.g., each array serves only one purpose)?				
	b. Calculate a/NM and enter score.				
SI.4(13)	a. How many units, are coded according to the required programming standard?				
	b. Calculate a/NM and enter score.				
SI.4(14)	Is repeated and redundant code avoided (e.g., through utilizing macros, procedures and functions)?				
SI.5(1)	a. How many data items are used as input? (total from all units)				
	b. Calculate 1/(1+a) and enter score.				
	c. Calculate b/NM and enter score.				
SI.5(2)	a. How many data items are used as output (total from all units)?				
	b. How many parameters in the units calling sequence return output values (total from all units)?				
	c. Calculate b/a and enter score.				
	d. Calculate c/NM and enter score.				
SI.5(3)	a. How many units perform a single, non-divisible function?				
	b Calculate a/NM and enter score.				

VALUE	YES	NO	NA
TOTALS			

DR =

SQ INPUT Assign 1 to all Y's answers and 0 to N's Count all answers other than Not Applicable (NA). Add all values and divide by the number of answers (not NA's). Assign to DR.

WORKSHEET 11B - STANDARDS REVIEW RESULTS (SR)

		VALUE	YES	NO	NA
UNIT NAME: _____					
Count number of executable lines of code. Set equal to	MLOC=				
MO.1(3)	Are the estimated lines of source code for this unit 100 lines or less, excluding comments?				
MO.1(4)	a. How many parameters are there in the calling sequence?				
	b. How many calling sequence parameters are control variables (e.g., select an operating mode or submode, direct the sequential flow, directly influence the function of the software)?				
	c. Calculate b/a and enter score.				
MO.1(5)	Is all input data passed into the unit through calling sequence parameters (i.e., no data is input through global area or input statements)?				
MO.1(6)	Is output data passed back to the calling unit (e.g., through calling sequence parameters or global areas)?				
MO.1(7)	Is control always returned to the calling unit when execution is completed?				
MO.1(8)	Is temporary storage (i.e., workspace reserved for intermediate or partial results) used only by this unit during execution (i.e., is not shared with other units)?				
MO.1(9)	Does this unit have a single processing objective (i.e., all processing within this unit is related to the same objective)?				
MO.2(5)	What is the cohesion value of this unit?				
SI.1(2)	Is the unit independent of the source of the input and the destination of the output?				
SI.1(3)	Is the unit independent of the knowledge of prior processing?				

		VALUE	YES	NO	NA
SI.1(4)	Does the unit description/prologue include input, output, processing, and limitations?				
SI.1(5)	a. How many entrances into the unit?				
	b. How many exits from the unit?				
	c. Calculate (1/a + 1/b)(* 1/2) and enter score.				
SI.4(1)	Is the flow of control from top to bottom (i.e., flow of control does not jump erratically)?				
SI.4(2)	a. How many negative boolean and compound boolean expressions are used?				
	b. Calculate 1-(a/MLOC) and enter score.				
SI.4(3)	a. How many loops (e.g., WHILE, DO/FOR, REPEAT)?				
	b. How many loops with unnatural exits (e.g., jumps out of loop, return statement)?				
	c. Calculate 1-(a/MLOC) and enter score.				
SI.4(4)	a. How many iteration loops (i.e., DO/FOR loops)?				
	b. In how many iteration loops are indices modified to alter the fundamental processing of the loop?				
	c. Calculate 1-(b/a) and enter score.				
SI.4(5)	Is the unit free from all self-modification of code (i.e., does not alter instructions, overlays of code, etc.)?				
SI.4(6)	a. How many statement labels, excluding labels for format statements?				
	b. Calculate 1-(a/MLOC) and enter score.				
SI.4(7)	a. What is the maximum nesting level?				
	b. Calculate a/MLOC and enter score.				

		VALUE	YES	NO	NA
SI.4(8)	a. How many branches, conditional and unconditional?				
	b. Calculate 1-(a/MLOC) and enter score.				
SI.4(9)	a. How many data declaration statements?				
	b. How many data manipulation statements?				
	c. Calculate 1-((b+c)/MLOC) and enter score.				
SI.4(10)	a. How many total data items, local and global, are used? (total from all units)				
	b. How many data items are used locally (e.g., variables declared locally and value parameters)?				
	c. Calculate b/a and enter score.				
SI.4(11)	a. Calculate 1-(DD/MLOC) and enter score.				
SI.4(12)	Does each data item have a single use (e.g., each array serves only one purpose)?				
SI.4(13)	Is this unit coded according to the required programming standard?				
SI.5(1)	a. How many data items are used as input?				
	b. Calculate 1/(1+a) and enter score.				
SI.5(2)	a. How many data items are used for output?				
	b. How many parameters in the units calling sequence return output values?				
	c. Calculate b/a and enter score.				
SI.5(3)	Does the unit perform a single, nondivisible function?				
SR INPUT:	Assign a value of 1 to all Y answers and a value of 0 to all N answers. Count the total number of answers (not NA's). Total score of answers and divide by number of answers. Assign to DR.	TOTALS			
		DR =			

☆U.S. GOVERNMENT PRINTING OFFICE 1988-[illegible]

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.

END

DATE

FILMED

5-88

DTIC